(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 March 2001 (15.03.2001)

PCT

(10) International Publication Number
WO 01/18656 A1

(51) International Patent Classification[7]: G06F 13/00, 15/00

(21) International Application Number: PCT/US00/24695

(22) International Filing Date:
8 September 2000 (08.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/392,995 9 September 1999 (09.09.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/392,995 (CON)
Filed on 9 September 1999 (09.09.1999)

(71) Applicant *(for all designated States except US)*: PERCUSSION SOFTWARE, INC. [US/US]; 92 Montvale Avenue, Stoneham, MA 02180 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: REYNOLDS, Barry [US/US]; 131 Grove Street, Wellesley, MA 02181 (US). IMRICH, Vernon, R. [US/US]; Apt. 2, 345 Cardinal Medeiros Avenue, Cambridge, MA 02141 (US). LANGLAIS, William [US/US]; 162 Orchard Street, Byfield, MA 01922 (US). HOWARD, Paul [US/US]; Apt. 3, 101 Hancock Street, Somerville, MA 02144 (US). GIAKOUMINAKIS, Anastasios [US/US]; Apt. 405, 200 Ledgewood Drive, Stoneham, MA 02180 (US).

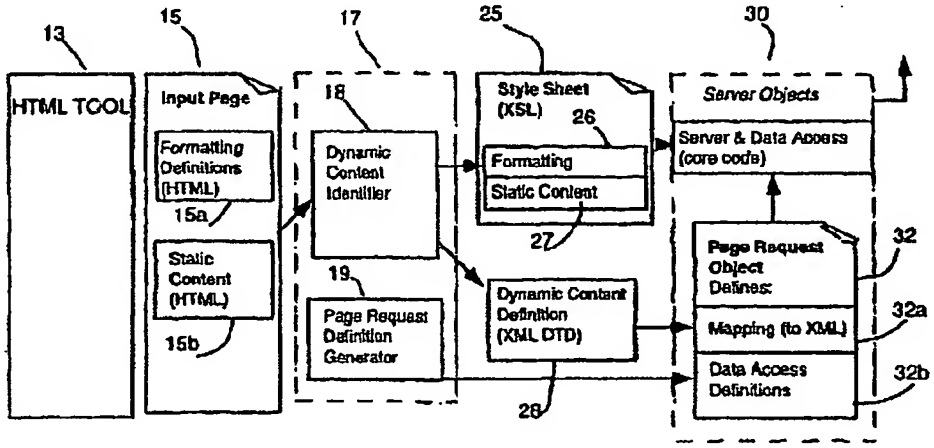
(74) Agent: STRETCH, Maureen; 26 Charles Street, Natick, MA 01760 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: VIRTUAL SERVER SYSTEM FOR DYNAMIC CONTENT IN WORLD WIDE WEB PAGES



(57) Abstract: A virtual server (54) which automates the retrieval of dynamic content for a web page and applies previously generated style (25) and content definition (28) sheets to format the content is disclosed. In the embodiment shown, an HTML document converted into an XSL style sheet document containing the static formatting and an XMLDTD document defining the dynamic content are created at design time. These identify which parts of the formatted document should be considered dynamic content and which should be considered as static. At run time, the invention automatically generates and serves virtual, hierarchically structured documents from structured data stored in a variety of data schemas. The style sheet (25) and content definitions (28) are applied to the data so that a virtual display document is created and sent in response to a user's request. The embodiment shown also allows data transformations to be defined visually. Data retrieval and processing are pre-compiled and optimized when virtual server process is started.

WO 01/18656 A1

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *With international search report.*

Description

Virtual Server System For Dynamic Content In World Wide Web Pages

Technical Field

The present invention relates generally to the field of creating interactive World Wide Web display documents for use over the Internet and more particularly to a virtual server for automatically generating web pages containing dynamic content.

Background

The explosive growth of the Internet and the World Wide Web (the Web) in the last few years has been accompanied by an ever-increasing use of the Internet and the Web for business and commercial purposes. However, most of the tools, such as the Standardized General Markup Language "SGML" and Hypertext Markup Language "HTML" languages and page generation tools which have been developed to allow individuals and companies to create Websites, assume that the content of a Website will be relatively static. HTML, for example, does not provide any syntax for dynamic content. For a retail website designed to display a retail catalog and price list that changes only once every few months, this tends not to be a problem, if products and prices do not change more rapidly. However, as more dynamic business applications are projected for the web, the need to transmit "live," current data through the website has increased dramatically. The proposed eXtensible Markup Language "XML" and eXtensible Style Language "XSL" formats should assist in standardizing the format for dynamic content but there are few, if any, tools which allow a statically defined website to handle

dynamic content changes readily. Consequently, those businesses seeking to develop such applications will typically need to hire skilled information technology (IT) resources to do so.

The illustration in Figure 2A(Prior Art) shows the different types of IT skill levels that this might entail. Presently, the simplest IT skill level is shown at level AA, where WYSIWYG (What You See Is What You Get) design tools such as DREAMWEAVER™ and FRONTPAGE™ provided by Microsoft, allow graphic designers to design web pages and to control the contents of them. Ideally, as level AA indicates, a web designer is likely to be able to use HTML (Hypertext Markup Language). This is a relatively simple, easy to use language that allows a web designer to "mark up" text and graphics in a web page so they will display as desired at the user's web browser. As the WYSIWYG term implies, WYSIWYG tools allow a designer to manipulate symbols and graphics in such a way that as the designer places them on a webpage, they can be viewed as they will appear to a user. For the typical static content of most websites these tools have eliminated the need for training the web designer in the details of IT and more advanced programming skills.

To the extent that the owner of a commercial website wishes to include the kind of dynamic content associated with many business applications, some higher level of IT skill will usually be required to accomplish this. As shown in Figure 2 (Prior Art), as the IT skill level needed ascends the pyramid shown, resources tend to become scarcer and usually more costly. This is true even with the advent of XML.

For example, if a user wants to develop an application that permits use of a volatile, rapidly changing internal database over the Internet, it is very likely that dynamic queries made against the database will turn up a variety of responses. HTML does not allow a designer to specify content as dynamic. Most present design tools have non-standard extensions to HTML that allow a designer to include some dynamic data. However, since these are non-standard extensions, the designer may be locked in to one tool. Also, they usually require additional training of the designer. Most problematic, however, is the fact that they are likely to intermingle data access design logic with display design.

These tools also may or may not be able to handle some of the more complex types of repeating fields in dynamic data. Problems frequently arise when the application under development attempts to map columns C1-C5, in a database, such as database 00 in Figure 3(Prior Art) to field F mappings in a web page 15, when the columns or the field contain hierarchical node components. Thus, F1 in webpage 15 may actually require several hierarchical sub -fields F1-F4 to display the hierarchy of subset data found in a column C in database 00. Conversely, in some cases a query made by a user against the database 00 may require the data access design to resolve repeated rows of data from database 00 into an aggregated document for display on the webpage 15. In order for such multiple results to be displayed on webpage 15, additional custom programming beyond the IT skill level of the typical web designer has usually been required.

The prior art tools usually put dynamic data directly into HTML formatted pages physically stored on the website, if they are able to handle dynamic data. Today,

many users would prefer to be able to put the dynamic data into XML format first, to take advantage of its tag labeling features. HTML only defines formatting --XML allows a user to classify the data using tags. For example, HTML only allows a user to format the appearance of the words "Charles Dickens", while XML allows a user to tag those words to indicate he is an author.

Present techniques for solving the IT problem, even with existing tools, usually involve having an IT professional who is skilled in Java or C++ or other languages work in serial with the web designer. Typically, the IT professional develops data access logic to provide the data for the web designer's page design. As shown in Figure 2B(Prior Art) this data access logic is usually embedded in the server enabled page 40 as a data access component 42 (an object or piece of code) or as a server side call 44 within a static HTML document. This means that the page design or "look and feel" of the website or application is tied together with data access and processing and vice versa. Page designers cannot re-design for look and feel without also affecting the data access design. Data access logic cannot be re-designed without also affecting the page design.

This also means designers and IT professionals must interact more than they might otherwise need to do. In normal developments this occurs serially. That is, a page designer may propose a layout which then has to be reviewed and accommodated by the data access IT personnel. If IT personnel think data access logic can be developed or modified to accomplish it, IT approves it and proceeds with development. If IT does not approve, several discussions may be needed to identify compromise solutions. Similarly, changes in the data access logic which

might necessitate changes in the page document display will have to be reviewed and accommodated by the graphics page designer. There can often be several iterations back and forth before final common specifications are agreed upon. This tends to cause delays and loss of productivity, as well as a greater chance for design errors.

With the interdependencies that exist between page design and data access design in this approach, other errors can arise easily based on misunderstandings between persons with different skill sets. What may seem to be an obvious page design requirement based on the data access logic design may not be obvious to a graphics designer handling the web page. Similarly, a page design that optimizes attractive display and ease of use may not appear so useful to the data access logic personnel if it makes their jobs more difficult. These discrepancies may not become apparent until well into the development cycle when the page design is first used in connection with the data access design.

Similarly, the business application being developed for the Internet may require that the web page designer and the data access designer allow a user to conditionally extract data for certain kinds of queries. In these cases, one field in the webpage may need to relate back to many columns of data in the database 00 backend or vice versa, depending on the data values or the context of the query. If the business wants to enable the web browser user to update a database 00, additional complications can arise. Is the update an addition, a deletion, or a change? How is the data sent to the web server handling the webpage and the database to be used? What formats are required? The web server protocol HTTP

used by designers of HTML documents operates on the principle of "request and return". This means web servers can deliver rich formatted information inside the web pages they serve, but they can only accept relatively simple information from within a request that is sent by a user at a web browser. Specifically, when a web server receives information, it natively accepts only a simple list of HTML parameters and their values. These parameters are either attached to the URL directly or are sent separately from the URL when submitted from a form using the POST method. Present attempts to address this problem with existing tools require even more special programming and exceptions.

For more complex business applications, automatic retrieval and processing of all information needed to make successive data access requests based on the results of the current request may be desired. In this kind of system, the application allows the user at a web browser to submit requests that "drill down" through multiple levels of data detail in the backend database 00 to find very specific data knowing only general information about its location. The problem faced by the data access IT person and the graphics person occurs when they try to make the system automatically determine the navigational information required about the location of the specific data, so that each successive request can be defined and altered independently of the other requests, but in such a way that the links between them are automatically maintained. For example, if the database contains a list of employees, and the user at a web browser only knows that she wants to find an employee named Joe, who lives in the Northeast, she might search the database for all employees named Joe. To find a specific one, the user might

request all those named Joe Smith. When the user sees the list she sees one living in Boston Massachusetts and one in Nashua, New Hampshire. If she remembers the area code, she can ask again to see the one with area code 603, which is the New Hampshire Joe Smith.Retrieving all the necessary information required so that a user can perform this kind of search is a significant challenge for present tools.

Recent industry projections indicate that the creation of a simple website with static displays can cost a company anywhere from several thousand US dollars to hundreds of thousands of dollars. Many business organizations may also have existing internal databases and applications programs they have spent thousands or millions of dollars developing.

At present, programming language tools for the Web, such as ASP™, provided by Microsoft and COLDFUSION™'s CFML™, provided by Allair, attempt to offer businesses a way to integrate the dynamic content of backend databases with statically defined website pages, but these tools involve using a specially developed proprietary language with its own complexity and hence, the need for a higher IT skill level. CFML, for example, has over 200 different commands a designer might need to learn. These tools as well as custom programmed efforts written in Java or C++ also tend to require the embedding of code in the webpage documents. This code is usually executed at run-time each time requests are made for dynamic data. As seen in Figure 2B(Prior Art) this can also slow down the overall execution of queries made at run-time over the Web.

Present attempts to design or modify web pages to incorporate dynamic data content are thus hampered by the potential need for higher, more expensive and scarcer IT skill levels, costly development times and delays, and final products that may be comparatively slower in execution time. Additionally, some current techniques require a user to include programs written in proprietary languages or custom-developed programs.

Summary

A virtual server system for generating dynamic content and a display document therefor to be sent over a network is disclosed which automates the retrieval of dynamic content and uses previously defined style and content definition sheets to format the display document. In the embodiment shown, dynamic data is accessed from a database, in any of a number of formats and schemas, using data access definitions and parameters generated at design time. The virtual server uses a previously defined XSL style sheet document containing the static formatting desired and an XML DTD document defining the dynamic content to be applied to the data to generate and serve the requested display document.

It is an aspect of the present invention that no physical display document is stored.

It is another aspect of the present invention that a display document can be generated at runtime by a virtual server which not only creates the document but also accesses any dynamic data needed in the display.

Brief Description of the Drawings.

Figure 1A is a block diagram of the present invention.

Figure 1B is a schematic drawing of the present invention in use.

Figure 1C is a block diagram of illustrative syntax used by the invention.

Figure 1D is a block diagram of illustrative syntax used by the invention.

Figure 1E is a block diagram of illustrative syntax used by the invention.

Figure 1F is a block diagram illustrating the formatting of repeating rows.

Figure 1G is a block diagram illustrating the formatting of nested tables.

Figure 1H is a block diagram illustrating the formatting of nested tables.

Figure 1I is a block diagram illustrating the formatting of repeating bulleted lists.

Figure 1J is a block diagram illustrating the formatting of repeating form elements.

Figure 2A(Prior Art) is a block diagram of skill levels required by the prior art.

Figure 2B(Prior Art) is a block diagram of prior art techniques.

Figure 3 (Prior Art) is a schematic diagram of a prior art access to a database, illustrating a problem.

Figure 4A (Prior Art) is a block diagram of server processing in the prior art.

Figure 4B is a block diagram of server processing using the present invention.

Figure 4C is a block diagram of the present invention in operation at runtime.

Figure 4D is a block diagram of a visual mapper of the present invention.

Figures 5 - 15 are flow and block diagrams of the present invention.

Detailed Description of the Invention

In Figure 1A an overview of the present invention is shown. An input page 15, which is being designed by a web designer, is depicted. Web pages such as input page 15 designed in standard HTML for the World Wide Web usually contain formatting definitions 15a and static content 15b. Working in conjunction with the web page designer's choice of designer tool 13, the present invention's dynamic content identifier (DCI) 18 evaluates the syntax used by the designer to create the page. Using a special, simplified syntax and context analysis discussed below, DCI 18 splits input page 15's elements into those which are static content and those which are dynamic content. In the embodiment shown, DCI 18 creates an XSL compliant style sheet 25, which describes the formatting 26 of the static content of input page 15. DCI 18 also creates a dynamic content definition 28, in XML compliant Document Type Definition (DTD) format. At design time a data access designer uses page request definition generator 19 of the invention to create data access definitions 32b, which are input parameters for the invention's page request object handler 32. When a web page such as input page 15 is requested from the present invention's virtual server (described below) at runtime, the invention's server applies style sheet 25 and dynamic content definition 28 to the content retrieved dynamically from a database to produce a fully formatted HTML page. This is performed by server objects 30, which include server and data access core code 31 and a page request object handler 32. Page request object handler 32 in the embodiment shown includes a handler routine which accepts parameters generated at design time by page request definition generator 19. These parameters define how to access the data in a database and how to map that data into XML format. The parameters also define how to connect to the database, how

to login, how to generate queries against it, and so on.

Despite the fact that HTML at present has no support for dynamic content, the present invention's dynamic content identifier (DCI) 18 allows designers to define all formatting for dynamic content using existing standard HTML tags, using any type of HTML editor or designer tool 13. In the embodiment shown, DCI 18 operates as a separate tool from designer tool 13. However, it could also operate as a subset or extension of a designer tool 13. DCI 18 recognizes specially prefixed field names and attributes, and contextual data to automatically generate an XSL compliant style sheet from the HTML input page 15.

Figure 1C illustrates some of the general types of specially prefixed field names and attributes. In the embodiment shown, a default prefix 80 such as "psx" can be used by the web designer to identify dynamic content. While the examples shown here use psx, another prefix could be supplied by the designer at the initial configuration of the system.

Still in Figure 1C, a web designer can use the default prefix 80 with any field names 82 that conform to the XML syntax. Note that these field names are written in HTML by the designer, but conform to XML syntax. Since XML field name syntax requires all fields to share a common "root" node name, the present invention automatically generates a "root" node name for all fields in the HTML document. The root node name comes from the base names of the HTML display document that is being analyzed by DCI 18. For example, assume that in Figure 1C, field 82c "psx-name/first" is contained in an input page 15 called person.htm.

In this example, the present invention gives field 82c the full name 84 of "person/name/first" and the associated DTD document 86 is called "person.dtd". In this example, the root node name is "person." Any field names 82 in the HTML document input page 15 which already use the correct root node name are not altered by DCI 18.

Turning now to Figures 1D, and 1E, two types of special syntax recognized by the present invention are illustrated. Figure 1D illustrates exposed syntax and Figure 1E illustrates hidden syntax.

First, in Figure 1D, the present invention enables the web designer to use the default prefix 80 with fields which are inserted directly inside existing HTML tags 91 or attributes 92. This is exposed syntax. That is, the field names 82 will show up on input page 15 during design time. This kind of syntax makes it easier for the designer to tell where the dynamic content is located, but makes it harder to get a sense of what the actual run-time page will look like. In addition, in exposed syntax, any dynamic links, images, or other page properties may appear broken to the designer until the actual run-time page is served. Generic tag content 90 illustrates the use of an exposed, specially prefixed fieldname 82 inserted between two tagnames 91. Generic tag attribute 92 shows use of a specially prefixed fieldname 82 as an attribute 92.

Figure 1E shows the use of hidden syntax. In this syntax, field names 82 are hidden as either generic tag content 100 or generic tag attributes 102. With hidden syntax, the designer is free to use "dummy" data throughout input page 15 for a

better sense of how the runtime document will look. At run-time, the actual values of the field names 82 will be supplied.

In most present browsers designed for the Worldwide Web, a browser does not send the web server the same information that it receives from a web server. When sending information, the browser sends only the information in a URL (Uniform Resource Locator) and a simple list of HTML parameters and their values. These are either attached to the URL or are sent separately along with the URL when submitted from a Form using the "POST" method. See Figure 4A(Prior Art).Thus, if a page has both queries and updates, the specially prefixed fields will be used by the present invention to populate data into web pages that are served. However, the browser at the user's location doesn't send back those same fields to the web server when it submits the form, because HTTP does not permit them. Instead, the browser only sends back the HTML parameter list based on the names defined in the HTML FORM. These parameter names are standard HTML and do not use or need the special prefix syntax of the present invention.

The present invention supports this situation by allowing one set of mappings to be active when the web page is served from a query, and a different set of mappings to be active when a form is used to update. On the query, the invention maps the specially prefixed fields. On a form update, the invention maps the HTML parameters defined by the form. The present invention also allows a query definition to set default values, attributes, and selection lists for all the form's fields, as well as the form's action attribute and other page properties. Other client programs, such as special Java application programs can send back

data in any format, including files, which can also be handled by the present invention. Usually, such special programs will use the same sets of fields as the input page.

Turning now to Figures 1F-1J, the syntax and context analysis used by the present invention to format repeating fields is illustrated. When a user is accessing dynamic data from a database, the same field in a document may be repeated in the display document multiple times, each time having a different value.

When the virtual server of the present invention (discussed in more detail below) creates an XML document from back-end data, (database 00) at run-time, it may repeat the same field in that document multiple times, each time with a different value. To format these repeated fields, the DCI 18 of the present invention generates, at design time, style sheets 25 capable of repeating both the fields and the surrounding HTML formatting tags.

DCI 18 also allows nesting of repeated fields and formatting within other repeating fields and formatting. A designer controls repeating formatting for fields based on placement of the fields in an HTML page and on the structure of the hierarchical node names shared by the repeated fields. DCI 18 and virtual server 54 of present invention analyze the prefixes and the context of these formats to determine the repeat patterns.

In Figure 1F, the method for repeating table rows is shown. In the embodiment shown, all fields share a common root node which associates all fields with their

parent document. Since this node is automatically part of all field names and designers do not usually enter it, it is not considered or shown in the examples below. In Figure 1F, a table row is repeated once for each time the first node (reading the field left to right) is shared by all repeating fields in the row. Any static text in the row repeats when the row repeats. All row and cell attributes (such as border and background color) also repeat.

In Figure 1F, table 114, only the second row 112 has a specially prefixed fieldname 82. The fieldname 82 has a single node N1 called here "itemfeature". The second row will therefore be repeated once for each occurrence of "itemfeature" in the run-time XML document.

Thus, as shown by table 116 of Figure 1f, if there is only one actual occurrence of an "itemfeature" (here the actual itemfeature is "It's cheap!) when the backend data from the database is served by server 54 in an XML format, then the example of row 120 will be displayed when the page is served. However, if the field has several entries in the database, the present invention allows the field formatting to be repeatedly applied to them, as seen in Table 118 , rows 122-126 of Figure 1F.

With reference now to Figure 1G, the present invention allows a web page designer to repeat the formatting for multiple fields sharing the same node. In table 130, as it appears at design time, row 132 has two static fields (Product Name and Product Features) and row 134 has two dynamic fields, as indicated by the special prefix 82, psx in this example. The formatting specified by the designer for these fields will be as seen at runtime in Table 140, when the dynamic data for

them has been retrieved and served. Here, the static row 142 is not repeated, but for each dynamic occurrence of a product name, one or more product features 146 are shown and formatted as originally specified by the web page designer. Each product appears on a new row and within that row the features are listed. It should be noted in these examples, that the formatting shown here which includes the prefix and node names is the formatting defined by the designer at design time - -in Figure 1G, that is table 130. The results - -here table 140 of Figure 1G- - are provided by virtual server 54 of the present invention in a display document in response to that formatting and to the actual results from database 00.

Turning now to Figure 1H, it can be seen that the embodiment shown also allows repeating of rows within rows. In Figure 1H , table 150 is defined at design time. In table 150, the outermost table row 152, repeats for the first shared node N1 (from left to right), the second table row 154 inward repeats for the second shared node N2 (continuing to right) and so on continuing inward until either all nested tables or shared nodes are exhausted. This formatting is described by DCI in an XSL style sheet and the dynamic data is identified by DCI 18 in a DTD format dynamic content definition 28. When dynamic data is retrieved at runtime about 3 different products158,162,164, each having one or more features170,172,174,176, the XSL style sheet 25 and the dynamic content definition 28 created at design time are applied by the invention's virtual server so that the information is displayed as shown in table 156 of Figure 1H . In Figure 1H note that the static content 160 "Take a look at these features!" is repeated with each occurrence of the product/name, while the changeable, dynamic content about products and features has been arranged as shown in Table 156.

All other repeatable formats nested within a table, such as the bulleted list shown in Figure 1I, , exhaust the shared nodes (from left to right) in the order of outermost to innermost repeatable formatting. Those skilled in the art will appreciate that other methods for analyzing the context and syntax could be used without deviating from the present invention. Special characters or indicators could be used, for example, or context might be analyzed differently.

Figure 1J illustrates how the DCI 18 of the present invention can be used to format forms from a browser, applying the same analysis.

Now turning to Figure 1B, virtual server 54 of the present invention is shown. In the embodiment shown, a standard web server 52, such as Microsoft Corporation's Internet Information Server (IIS) web server is used, and Microsoft's standard Windows NT™ operating system 50 is used as well. As will be apparent to those skilled in the art, other web servers, operating systems, and network protocols could be used without deviating from the spirit of the present invention. Figure 1B illustrates the relationship of the invention's design time tools --data access page request definition generator 19 and web page designer's DCI 18-- with actual runtime programs, such as the present invention's virtual server 54, and other web applications 60, web browser 70, and database 00.

Turning briefly to Figure 4B, an overview of the operation of virtual server 54 of the present invention is shown. When a query request is made from a user at UU, it is translated by the web browser into standard http protocol requests S5, such as

GET URL, POST Body, and forwarded to the webserver machine S15. There the webserver 52 maps the URL to find the predefined request object handler 32 of the present invention's virtual server 54. Predefined request object handler 32 uses the dynamic content definition 28 created at web page design time in DTD format, and data access definitions 32b created at data access design time to find a database 00, to get the data at block 25, and then to generate the reply in XML format that will be combined by virtual server 54 with the previously defined XSL style sheet to create a web page display document DD that has all the dynamic fields requested and formatted as the web page designer intended.

In the embodiment shown, virtual server 54 automatically generates and serves virtual, hierarchically structured documents from structured data stored in a variety of data schemas (relational, tabular, indexed, hierarchical) in a database 00, In addition, all data transformations necessary to map from the storage schema of database 00 into the front-end display document DD hierarchy schema may be defined visually without the need for any code, (as described below) thus simplifying the data access logic as well. The data retrieval and data processing logic are pre-compiled and optimized when the virtual server 54 process is started. No run-time code is generated nor is any code interpreted while processing run-time requests. The transformation mappings done by virtual server 54 allow structured data organized in rows and columns to be automatically repeated or collapsed as necessary to fit a hierarchical document's content structure.

In the embodiment shown, no physical input page 15 was actually stored on disk at design time by DCI 18. Instead, an input web page 15 in the present invention

is a virtual document described by XSL style sheet 25 and dynamic content definition 28. Consequently, in the present invention, a request for such a virtual document appears the same (uses the same request syntax) as a request for an actual physical document being served from disk, although the document in the case of the present invention is entirely virtual. Because the document is virtual, no physical web page document need be stored on, or requested from disk in order for the final display document DD to be served.

In other words, while prior art approaches usually store a physical document - - such as input page 15- - containing HTML formatting and embedded code , the present invention's virtual server 54 generates a virtual document at runtime, based on the parameters stored in the predefined request definition object data access definitions 32b and XSL style sheet 25.

Virtual server 54 of the present invention both generates and serves XML and HTML documents via the HTTP protocol from various back end data sources- - database 00 - -including relational SQL (Standard Query Language) databases, other XML or HTML files, hierarchical databases (AS/400), and file systems, among others. This allows the data access logic and design to be completely separated from the "look and feel" of the web page design. Returning to Figure 1A for a moment, page request definition generator 19 is used by the data access designer to create the data access definitions 32b that are stored in page request object data access definitions 32b.

Now referring to Figure 4C, when a request R is made by a user UU, it travels

using standard HTTP protocol over the Internet II to computer S15, which is running an operating system 50, a webserver 52 and the present invention's virtual server 54. Virtual server 54 executes the page request object handler 32 which is associated with the fields in the request R, and uses the data access definitions 32b, which were created by the page request definition generator 19, to access database 00 and find data 01 which meets the request R. The data access definitions 32b can be defined to work with data 01 in a number of common formats, or schemas, such as relational databases, flat fields databases and so on. The data access definitions 32b are used to find the database 00 and make the appropriate read or write requests against it.

In the embodiment shown in Figure 4C, virtual server 54 maps the data 01 found into XML format using the dynamic content definition 28 created by DCI 18 at web page design time. As part of this processing, the formatting defined in style sheet 25 is applied to both the dynamic and the static content of the data found and the virtual display document DD being created. When formatting is completed, virtual web page display document DD is sent back to the user UU over the Internet II, using the standard webserver software 52. As seen in this example, display document DD is not stored at the webserver site or in the database 00 but is created dynamically in accordance with the parameters that are stored in page request object XML mappings 32a and data access definitions 32b, XSL style sheet 25 and dynamic content definition 28. In the embodiment shown, page request object handler 32 includes handler code which accepts the information in the data access definitions 32b, dynamic content definition 28 and XSL style sheet 25 as parameters to be used in satisfying the request R. Thus, no

code is embedded in display document DD. This makes it easy for the data access logic to be changed without affecting the web page formatting and vice versa.

In addition, the mapping to XML format performed by the invention 's virtual server 54 is also simplified by use of the current invention's visual mapper 33, shown in Figure 4D. As seen there, the data access designer can use visual mapper 33 to specify how data 01 found in database 00 should be mapped into XML fields and formats stored in XML mapping 32a. Since this mapping is done at data access design time, it turns the actual mapping at runtime into the application of predefined parameters.

Turning back briefly to Figure 1G, an example of XML 141 generated by page request object handler 32 in virtual server 54 of the present invention is shown. When page request object handler 32 applies the XSL style sheet 25 formatting, and the dynamic content definition 28 at runtime, the results will appear as table 140 when that particular request is handled.

With reference now to Figure 1A again, page request definition generator 19 of the present invention is used by the data access designer, at design time to create data access definitions 32b of the page request object handler 32. Thus, at run time, page request object handler 32 will have as input parameters the XML mapping resulting from dynamic content definition 28 and the data access definitions 32b, as well as XSL style sheet 25.

This enables virtual server 54 of the present invention to take advantage of the

dynamic content definition 28 to determine what the output format should be. Dynamic content definitions 28 can also contain information regarding the repetitiveness of the data elements. For instance, if there is a table in database 00 called orders with the following structure:

order_id	Item
123	1
123	2
123	3
456	1
456	2

there are many ways to create an XML document from this data. One way to force the structure is to build the dynamic content definition 28 in the appropriate manner. For example:

```
<!ELEMENT Orders  (Order*)>
<!ELEMENT Order   (ItemList)>
<!ATTLIST Order
 id  ID    #REQUIRED>
<!ELEMENT ItemList (item*)>
<!ELEMENT item    (#PCDATA)>
```

This has Orders as a root node which contains Order elements. Looking at the above dynamic content definition 28, there is an asterisk next to Order in the Orders element declaration. This indicates that Order is a repeating entry. Order contains a single attribute, the order id. That maps to an order_id column. There is also an ItemList element which contains item elements. The item elements map to

the item column. Once again, from the dynamic content definition 28 it is seen that item is a repeating entry. The id attribute, on the other hand, cannot repeat, because attributes cannot repeat in XML. Based upon these relationships, the present invention makes a few determinations:

- -the id attribute of the Order element must be the identifier for item entries. This means that for each matching id (order_id in the back-end) the invention creates one ItemList entry. It then collapses multiple item entries into their matching ItemList entry.

Since the id attribute (order_id column) is the key, the back-end query must be sorted on order_id. If "Use WHERE table" is specified in the data selector on the query pipe and no sorted columns have been specified, the present invention will define orderings (ORDER BY) in the SQL statement issued to the back-end database 00. If sorted columns have been specified, or "Manually enter SQL" has been chosen, the user must be sure to properly sort the data. In the above example, the user wants to collapse items based upon their order id. Consequently, the invention will sort on the order_id column.

This would cause the following XML to be generated:

```
<Orders>
    <Order id="123">
        <ItemList>
            <item>1</item>
            <item>2</item>
```

```
                <item>3</item>
            </ItemList>
        </Order>
        <Order id="456">
            <ItemList>
                <item>1</item>
                <item>2</item>
            </ItemList>
        </Order>
</Orders>
```

If a slight change is made as follows, to the dynamic content definition 28 :

```
<!ELEMENT Orders  (Order*)>
<!ELEMENT Order   (item)>
<!ATTLIST Order
  id   ID    #REQUIRED>
<!ELEMENT item   (#PCDATA)>
```

once again Orders is a root node which contains Order elements. Order contains the order id, but now it contains the item element directly. This structure indicates there are repeating Order elements, but id and item are now a one to one match. For this model, the invention makes no assumptions about sorting requirements. It would now generate the following XML:

```
<Orders>
    <Order id="123">
        <item>1</item>
    </Order>
    <Order id="123">
        <item>2</item>
    </Order>
    <Order id="123">
        <item>3</item>
    </Order>
    <Order id="456">
        <item>1</item>
    </Order>
    <Order id="456">
        <item>2</item>
    </Order>
</Orders>
```

Further complicating query mappings, the data may contain conditionals which can change the way data is mapped. For example, assume the data contains a phones table as follows:

phone_id	cont_id	phone_type	phone_no
1	1	H	111-1111

2	1	B	222-2222
3	1	F	333-3333
4	2	B	444-4444
5	2	F	555-5555

and the following dynamic content definition 28 is generated:

```
<!ELEMENT Contacts (Contact*)>
<!ELEMENT Contact  (Phones)>
<!ATTLIST Contact
 id   ID    #REQUIRED>
<!ELEMENT Phones  ((home | business | fax)*)>
<!ELEMENT home    (#PCDATA)>
<!ATTLIST home
 id   ID    #REQUIRED>
<!ELEMENT business (#PCDATA)>
<!ATTLIST business
 id   ID    #REQUIRED>
<!ELEMENT fax     (#PCDATA)>
<!ATTLIST fax
 id   ID    #REQUIRED>
```

In this example, first note that the Contacts/Contact element is repeating. It's id attribute maps to the cont_id column. This tells the present invention it will be sorting on cont_id and collapsing phone records into the Phones element. The pipe character in the Phones element declaration signifies that the Phones element contains home, business or fax elements. Furthermore, the asterisk signifies that there may be many of these. To get the phone numbers into the appropriate places, the data access designer builds a mapping such as the following:

XML Field	Back-End Column	Conditionals
Contacts/Contact/@id	cont_id	

Contacts/Contact/Phones/home/@id	phone_id	phone_type = H
Contacts/Contact/Phones/home	phone_no	phone_type = H
Contacts/Contact/Phones/business/@id	phone_id	phone_type = B
Contacts/Contact/Phones/business	phone_no	phone_type = B
Contacts/Contact/Phones/fax/@id	phone_id	phone_type = F
Contacts/Contact/Phones/fax	phone_no	phone_type = F

As the invention processes the data, in the embodiment shown, it will first create the Contacts/Contact element and set the id to the cont_id column value. For each matching cont_id value, it will then look at the conditionals phone_type column value to determine what to do. If the column contains 'H' as it's value, it will create a Contacts/Contact/Phones/home element containing the phone_no column's value and set the id attribute to the phone_id column's value. If the phone_type column contains 'B' or 'F' it will create the business or fax element in the same fashion. Based upon the data specified above, the following XML document is generated:

```
<Contacts>
    <Contact id="1">
        <Phones>
            <home id="1">111-1111</home>
            <business id="2">222-2222</business>
            <fax id="3">333-3333</fax>
```

```
        </Phones>
    <Contact>
    <Contact id="2">
        <Phones>
            <business id="4">444-4444</business>
            <fax id="5">555-5555</fax>
        </Phones>
    <Contact>
</Contacts>
```

When performing updates, inserts or deletes, the embodiment of the invention shown here needs an attribute or element to be defined which tells it which of these three actions it should take.

Unlike query mappings which take advantage of the dynamic content definition 28, update mappings rely heavily upon the XML mapping definitions 32a and the input data structure. For example with the following XML document:

```
<Contacts>
    <Contact id="1">
        <Phones DBActionType="UPDATE">
            <home id="1">111-1111</home>
            <business id="2">800-222-2222</business>
            <fax id="3">333-3333</fax>
        </Phones>
    <Contact>
    <Contact id="2">
        <Phones>
            <business id="4">444-4444</business>
            <fax id="5">555-5555</fax>
        </Phones>
    <Contact>
<Contacts>
```

there is an added attribute, DBActionType, to the Phones element. When performing updates, inserts or deletes, the page request object handler 32 of the present invention needs an attribute or element to be defined which tells it which of these three actions it should take. The system would then see a need to update the phone numbers for the first contact, but skip the second contact.

This entails applying data from multiple XML fields into the same back-end table. To do this, the data access designer has created a set of mappings for each XML group to its corresponding back-end group. By repeating the back-end column entries, the present invention will process each group of entries as a separate update to the back-end database 00.

Enabling users to navigate, link, or "drill down" from one page to the next is also provided for by the present invention. In its simplest form, the data access designer can simply use a request URL on the "source" (current) page to point to the next successive "target" page. However, in a dynamic environment things quickly get more complicated. The request URL to the"target" page needs to supply more information in order to get the right type of dynamic content.

For example, in a static environment with only one person, the request URL needs to only specify "person.htm" but in a dynamic content environment with many people, the URL would need to specify "person.htm?firstname=Joe&lastname=Smith" in order to get the exact person page for "Joe Smith".

Complicating issues further, the source page may itself have been created dynamically based on some other request. Suppose for example, the source page contained a list of people to choose from by name, but now the target page required an employee ID number to locate any specific person such as, "person.htm?employeeid=123". Here, though the source page is designed to show people by name, it would have to "know ahead of time" that it is linked to a target page which requires the employee ID number and not the person's name. Otherwise, the source page would be delivered without the employeeID and it would not be possible to link it to the target page. Furthermore, if the target page is changed to use something other than employee ID, then the source page would have to be kept in synch or the links could be broken.

The present invention solves these problems through the use of navigation links. As seen in Figure 14, the data access designer visually links the "source" data request 1400 to all the other possible "target" data requests, 1402,1404,1406 etc., that could be linked to from that source. When this is done, the "source" request handler auto-generates all the possible URLs required to link from the source web page to the target pages. Each of these URLs are placed in the "source", in addition to the other data defined by that source request. If a linked target resource definition is altered to require a new type of request URL, the source request handler is also altered so as to generate the new URL to the target resource.

These autogenerated URLs may be used however the designer wishes in the

actual

pages returned. Most often, the URL is placed as the contents of the HREF attribute on the A (anchor) tag to allow for hypertext linking between documents. However, it is also commonly used in the SRC attribute of the IMG tag (for navigation links attached to non-text resources), or the ACTION attribute of the FORM tag (for navigation links attached to update resources), or anywhere else a standard URL is used in an HTML page or other client application.

When a "source" request is linked to a "target" request at design time, the subsequent request handler for the "source" is created with additional properties as follows for each linked target. In the embodiment shown, the "source" request must always be a query.

1. The target request definition is examined to determine any required request parameters.

2. If the target is a query:

a. The data selection properties for the target are determined.

b. Required data for selection is mapped to request parameters for the URL

c. The source request handler is updated to include processing of the additional parameters into valid URL syntax.

3. If the target is an update the links may be defined to allow updates, inserts, or deletes. There are two cases to be considered:

a. URLs for inserts

i. Only the request properties for the target are mapped to request

parameters for the URL.

ii. The source request handler is updated to include processing of the additional parameters into valid URL syntax.

b. URLs for updates or deletes

i. The data update properties for the target are determined.

ii. Required key data for the update is mapped to request parameters for the URL

iii. The source request handler is updated to include processing of the additional parameters into valid URL syntax.

At run time, the request handler for the "source" page now includes the necessary mechanisms to generate all the URLs to each linked target. When the source page is processed, the additional URL generation takes place, and the URLs are included with the source document when it is served.

Detailed flow diagrams illustrate the logic of the embodiment shown. Beginning with Figure 15, DCI 18's processing of the HTML nodes is shown. An input page 15, here shown as HTML 1500 is analyzed HTML node by HTML node, as illustrated at step 1502. Once an HTML node is selected, DCI 18 checks to see if it has child nodes at decision block 1504. If it does, DCI 18 loops through each HTML childnode until the last or innermost is found. When that is found, processing proceeds at step 1510, to process all the HTML node information from static text at 1512, to other attributes at 1538. The prefix and context analysis described above is applied to each, as appropriate. Then DCI 18 corrects the node context at step 1540, transforms the HTML node to an XSL node at 1510, and replaces the older

one, if any. Processing continues in this manner at 1506 for each HTML node in the child node list. When processing is completed, an XSL style sheet 25 and a dynamic content definition 28 have been created.

Turning now to Figure 5, initiation of the invention's virtual server 54, here referred to as E2 is shown. At step 500 the webserver hook of virtual server 54 receives a request. At decision block 502 it checks to see if the URL specified in the request is in this server's root. If it is not, the event is released untouched at step 504. If the request does have a URL contained in virtual server 54's root, the invention checks to seek if there is at least on application also in the root, and if the rest of virtual server 54 is up and running. If not, the event is either released at step 508, or the checks for maintenance processing at step 512 through 522 are executed. If virtual server 54 is up, the request is handed to its dispatcher at step 520. At step 524 the input parameters of the request are parsed and then run through virtual server' 4's request filters, shown below. Checks are made to see if the request is tied to an application, or if SSL (Secure Socket Layer or other encryption technology is required) and appropriate reports are generated. At step 542, the invention can check use statistics to select the most available server for handling the present request and forward the request to that server.

Figure 6 is a flow diagram which illustrates some of the pre-processing of the request done by the present invention. At decision block 602, the system checks to see if raw input data exits have been defined by the user, and if they have, an exit handler is executed for each one specified at step 610. Exits, as they are defined in the embodiment shown, are custom code a data access designer may have

written. While it is a goal of the present invention to minimize the need for any custom code, it also permits such code to be included by a data access designer, if desired. If no special exits are need, the invention proceeds to step 604 to map the input data in the request to the back-end data in database 00, by mapping names, and converting data types as necessary. If this succeeds, processing flows to the statement builder at step 618.

The statement builder of the present invention is shown in Figure 7. Essentially, the statement builder determines what kind of data access is required. As seen at block 710, if a simple query is required, processing proceeds to execute the query at step 718. If an insert is requested, that processing is selected, and so on. Figure 8 illustrates the processing to execute a query, Figure 10 illustrates the logic for inserting data, Figure 12 illustrates update processing, and Figure 13, delete processing. Figure 11 shows the processing of an actual transaction against database 00, and Figure 9 illustrates the generation of the results. As seen at step 926 in Figure 9, the results are the generated HTML display document DD.

In the embodiments shown, Windows NT- compatible personal computers and workstations are used to implement the present invention. However, those skilled in the art appreciate that other computer platforms such as SUN Microsystems workstations, or IBM Mainframes could be used as well, either in client server or centralized or distributed network configurations. While the embodiments shown are programmed using C++, other programming languages can be used as well. Similarly, while software has been used along with the above computer systems

to implement the invention, part or all of the invention could be included in firmware or hardware circuitry without deviating from the spirit of the present invention.

Claims

What is claimed is:

1. A method for generating a virtual display document containing dynamic content over a network in response to a request, comprising the steps of:

accessing dynamic data in a database using a previously generated dynamic data access definition;

applying a previously generated style sheet to format the static fields of the virtual display document;

applying a previously defined dynamic content definition for the virtual display document which describes how dynamic fields are to be formatted; and

dynamically serving a virtual display document formatted accordingly in response to the request.

2. The method of Claim 1, wherein the step of accessing dynamic data in a database using a previously generated dynamic data access definition further comprises the step of using the data access definition to find the database and make the appropriate input and output requests to the database.

3. The method of Claim 1 wherein the step of accessing dynamic data in a database using a previously generated dynamic data access definition further comprises the step of using the previously generated dynamic data access definition to work with data in any of a plurality of common formats.

4. The method of Claim 1, wherein the step of dynamically serving a virtual display document further comprises the step of mapping data access definitions,

style sheets and dynamic content definitions to generate the virtual display document.

5. The method of Claim 4, wherein the step of mapping data access definitions, style sheets and dynamic content definitions further comprises the step of mapping data access definitions, style sheets and dynamic content definitions into XML format.

6. The method of Claim 4, wherein the step of mapping data access definitions, style sheets and dynamic content definitions is performed dynamically by a page request object handler.

7. The method of Claim 6, wherein the step of mapping data access definitions, style sheets and dynamic content definitions further comprises the steps of:

- - mapping the correspondences between data access definitions, style sheets and dynamic content definitions and XML fields and formats at design time and storing the mapping results in the database; and

- -using the page request object handler to apply the mapping results when the virtual display document is served.

8. The method of Claim 7 wherein the step of mapping the correspondences is done by a visual mapper.

9. An apparatus for generating a virtual display document containing dynamic content over a network in response to a request, comprising:

means for accessing dynamic data in a database using a previously generated

dynamic data access definition;

means for applying a previously generated style sheet to format the static fields of the virtual display document;

means for applying a previously defined dynamic content definition for the virtual display document which describes how dynamic fields are to be formatted; and

means for dynamically serving a virtual display document formatted accordingly in response to the request.

10. The apparatus of Claim 9, wherein the means for accessing dynamic data in a database using a previously generated dynamic data access definition further comprises means for using the data access definition to find the database and make the appropriate input and output requests to the database.

11. The apparatus of Claim 9 wherein the means for accessing dynamic data in a database using a previously generated dynamic data access definition further comprises means for using the previously generated dynamic data access definition to work with data in any of a plurality of common formats.

12. The apparatus of Claim 9, wherein the means for dynamically serving a virtual display document further comprises means for mapping data access definitions, style sheets and dynamic content definitions to generate the virtual display document.

13. The apparatus of Claim 12, wherein the means for mapping data access

definitions, style sheets and dynamic content definitions further comprises means for mapping data access definitions, style sheets and dynamic content definitions into XML format.

14. The apparatus of Claim 12, wherein the means for mapping data access definitions, style sheets and dynamic content definitions is performed dynamically by a page request object handler.

15. The apparatus of Claim 14, wherein the means for mapping data access definitions, style sheets and dynamic content definitions further comprises:

means for mapping the correspondences between data access definitions, style sheets and dynamic content definitions and XML fields and formats at design time and storing the mapping results in the database; and

means for using the page request object handler to apply the mapping results when the virtual display document is served.

16. The apparatus of Claim 15 wherein the means for mapping the correspondences is a visual mapper.

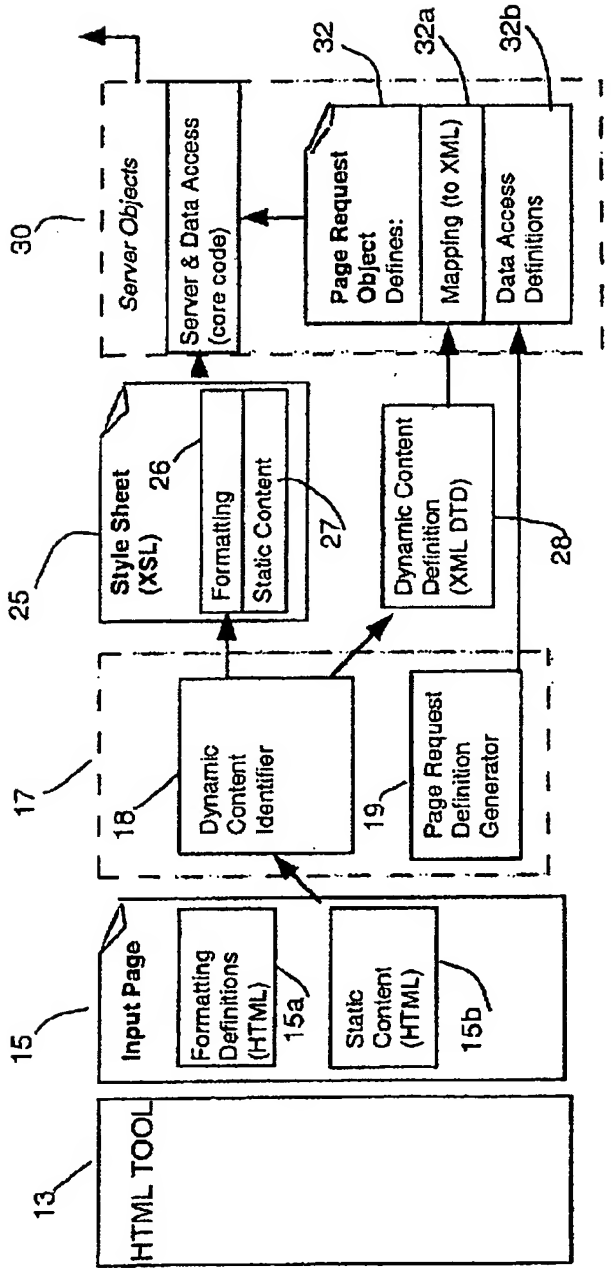

HTML TOOL
13
Input Page
Formatting Definitions (HTML)
15a
Static Content (HTML)
15b
15
17
Dynamic Content Identifier
18
Page Request Definition Generator
19
Style Sheet (XSL)
25
Formatting
26
Static Content
27
Dynamic Content Definition (XML DTD)
28
Server Objects
30
Server & Data Access (core code)
Page Request Object Defines:
32
Mapping (to XML)
32a
Data Access Definitions
32b

Fig. 1A

Fig. 1B

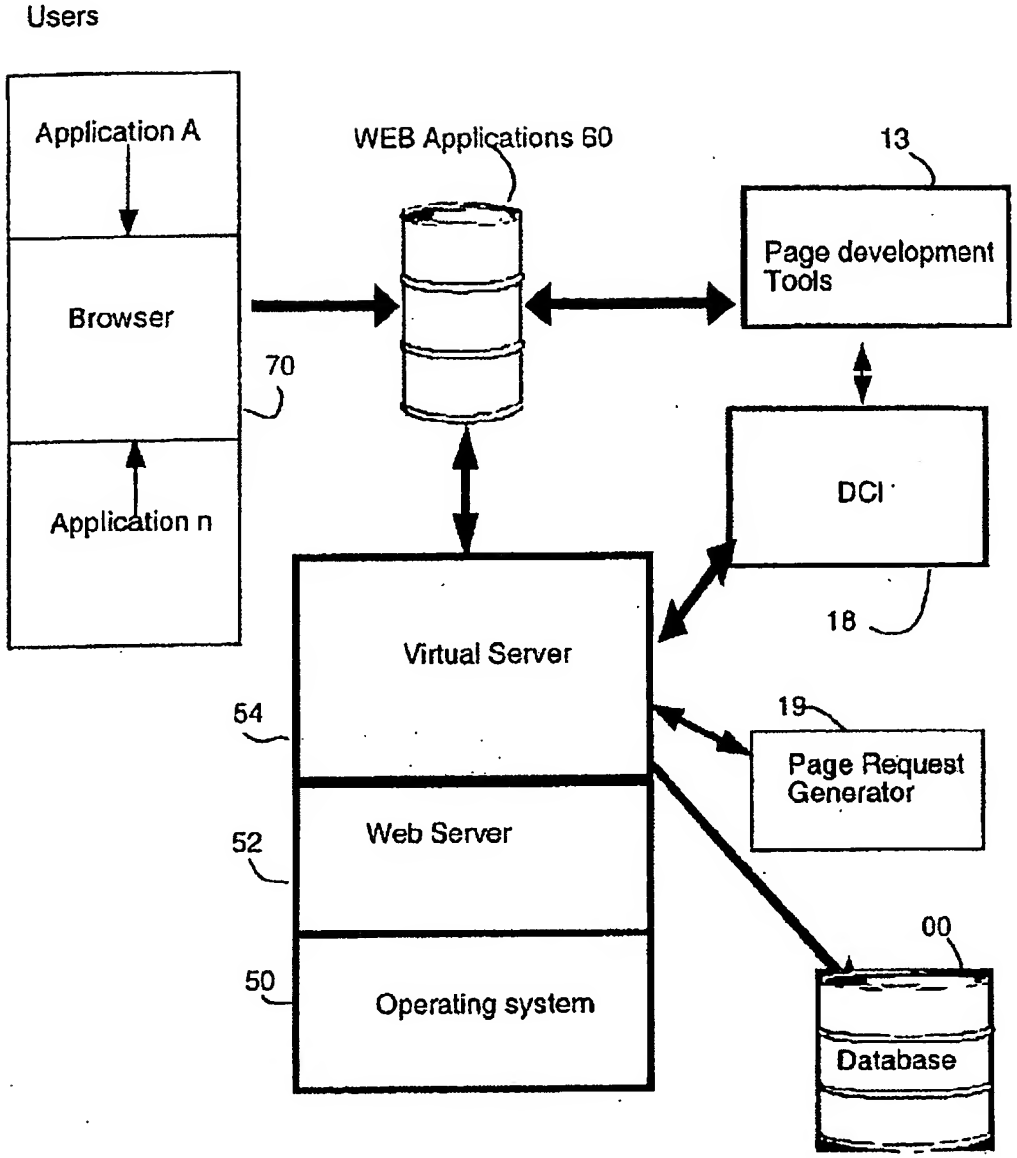

Users
Application A
Browser
Application n
70
WEB Applications 60
13
Page development Tools
DCI
18
Virtual Server
54
Web Server
52
Operating system
50
19
Page Request Generator
00
Database

Fig. 1C

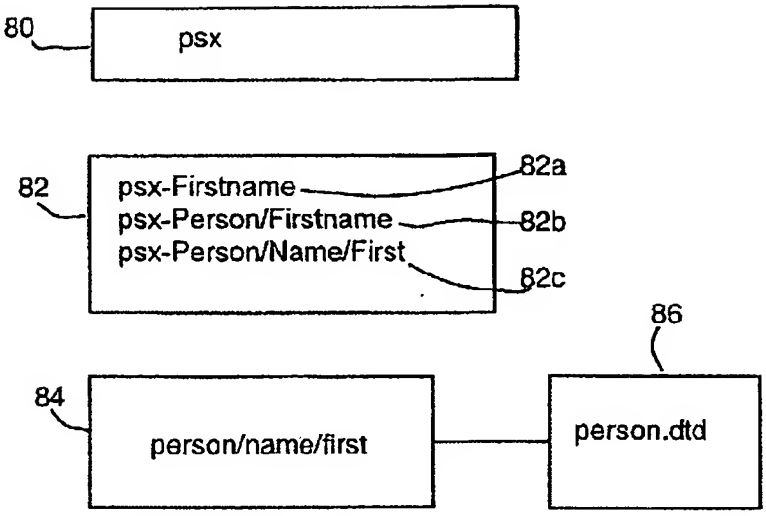


Fig. 1D

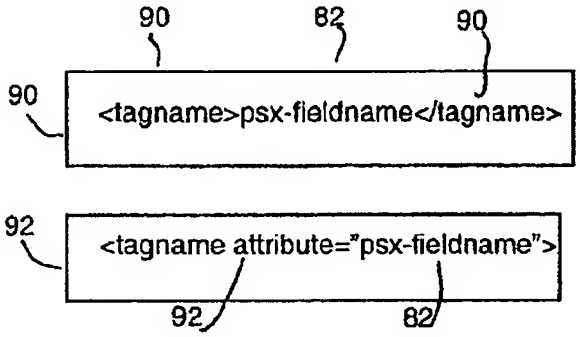


Fig. 1E

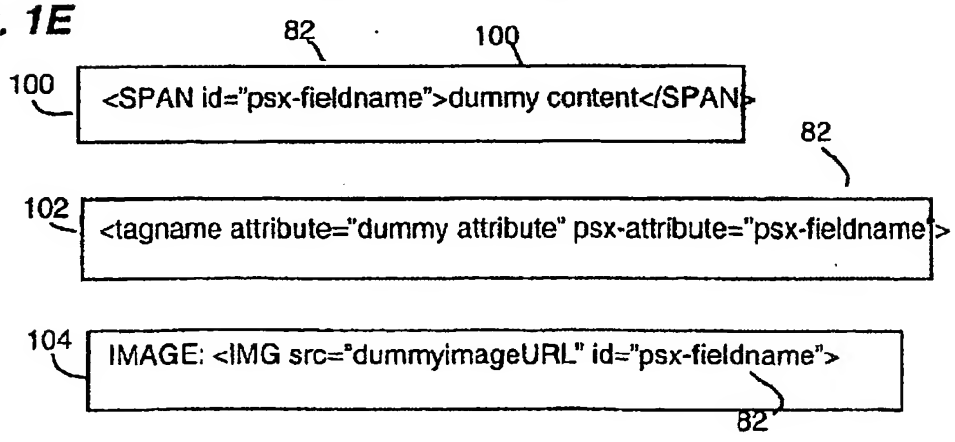


Fig. 1F

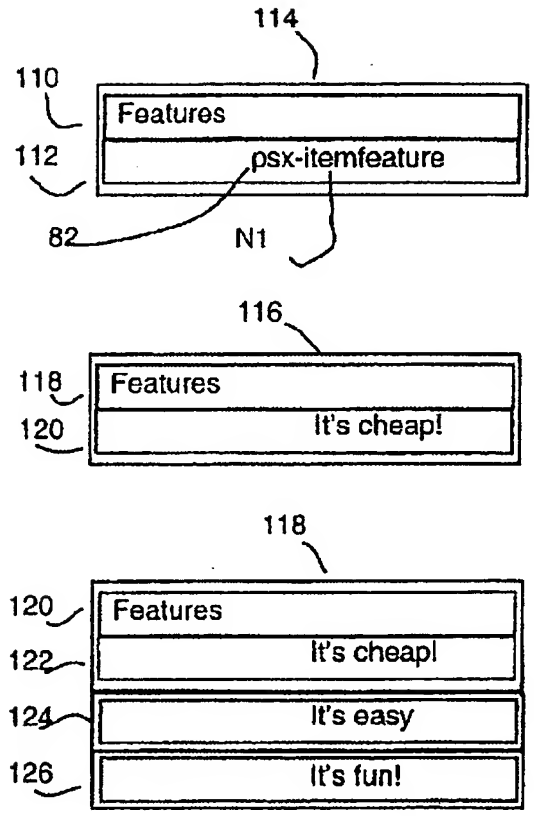
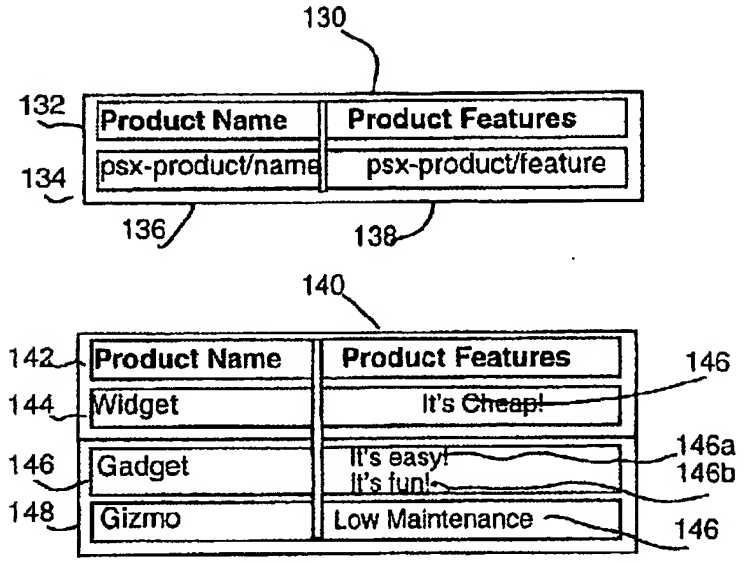

114
110
Features
112
psx-itemfeature
82
N1
116
118
Features
120
It's cheap!
118
120
Features
122
It's cheap!
124
It's easy
126
It's fun!

Fig. 1G


130
132
Product Name
Product Features
134
psx-product/name
psx-product/feature
136
138
140
142
Product Name
Product Features
146
144
Widget
It's Cheap!
146
Gadget
It's easy!
It's fun!
146a
146b
148
Gizmo
Low Maintenance
146

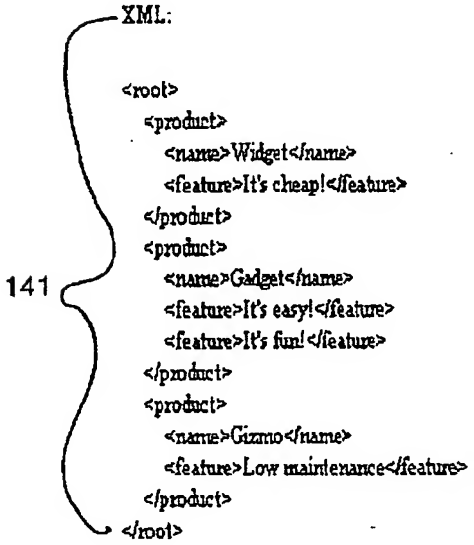

XML:
<root>
<product>
<name>Widget</name>
<feature>It's cheap!</feature>
</product>
<product>
<name>Gadget</name>
<feature>It's easy!</feature>
<feature>It's fun!</feature>
</product>
<product>
<name>Gizmo</name>
<feature>Low maintenance</feature>
</product>
</root>
141

Fig. 1H

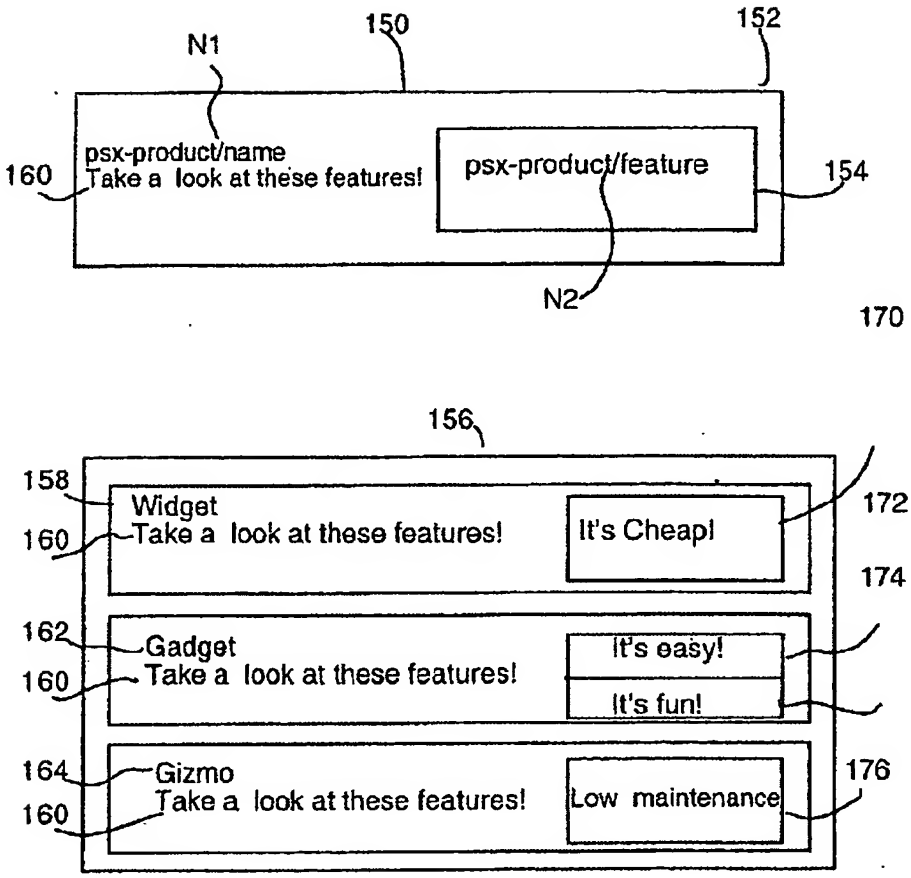

N1
150
152
160
psx-product/name
Take a look at these features!
psx-product/feature
154
N2
170
156
158
Widget
160
Take a look at these features!
It's Cheap!
172
174
162
Gadget
160
Take a look at these features!
It's easy!
It's fun!
164
Gizmo
160
Take a look at these features!
Low maintenance
176

Fig. 1I

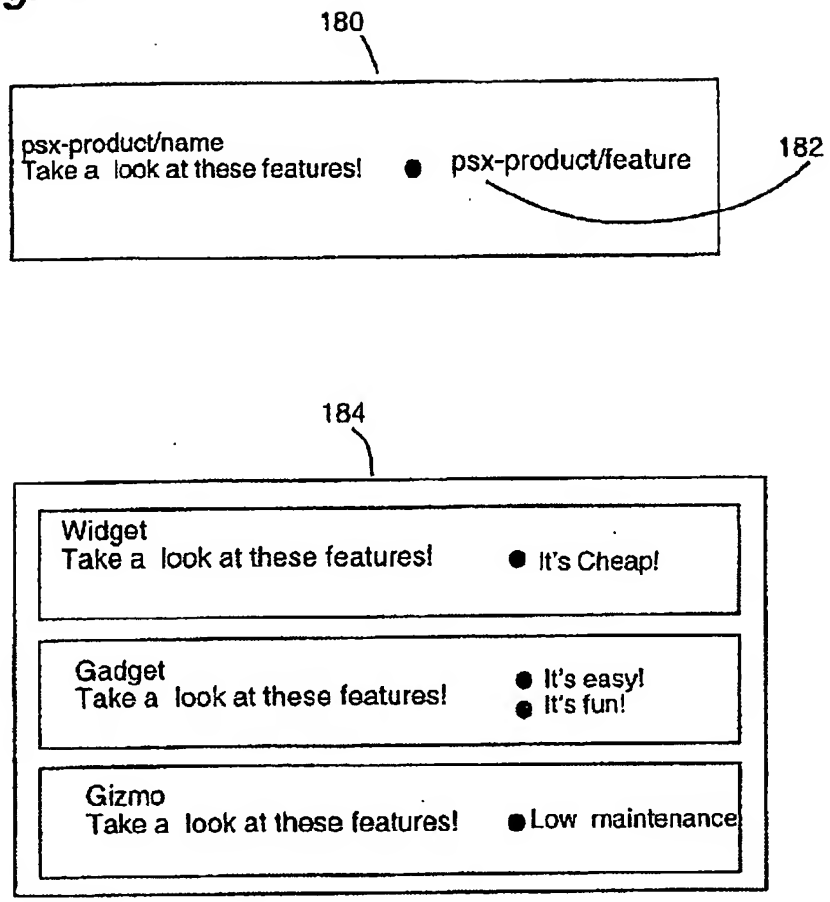

180
psx-product/name
Take a look at these features!
psx-product/feature
182
184
Widget
Take a look at these features!
It's Cheap!
Gadget
Take a look at these features!
It's easy!
It's fun!
Gizmo
Take a look at these features!
Low maintenance

Fig. 1J

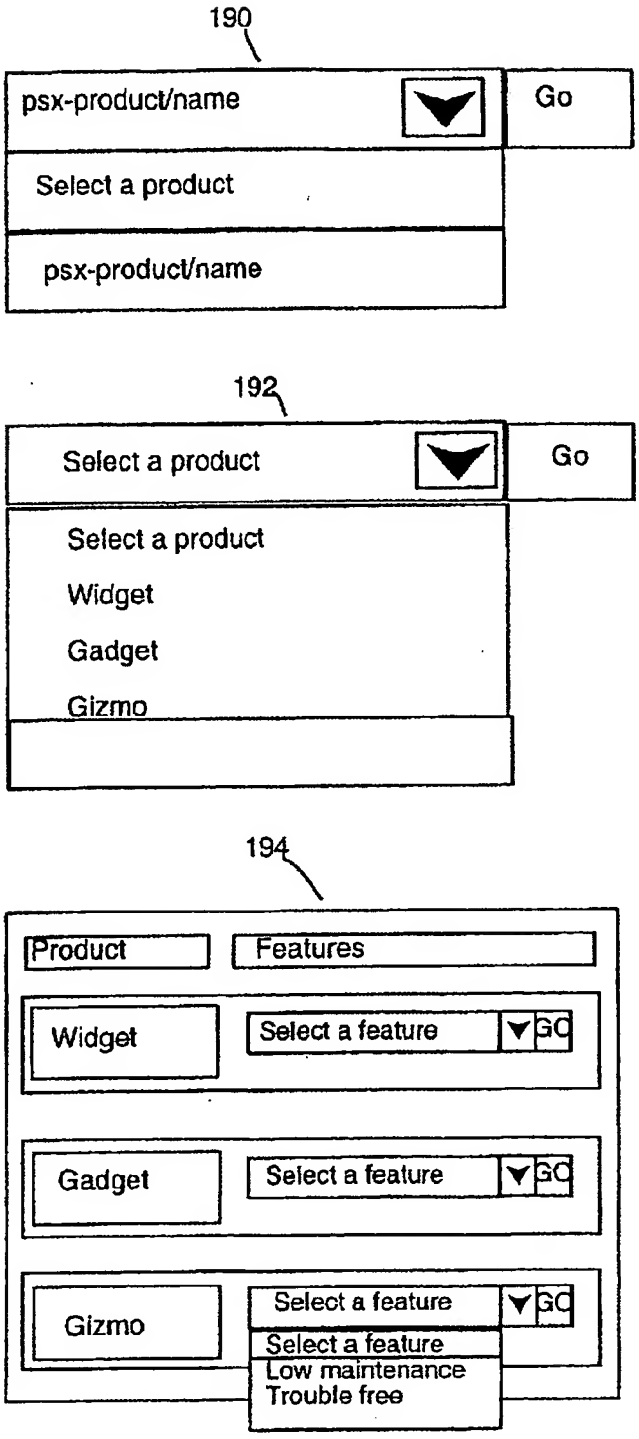

190
psx-product/name
Go
Select a product
psx-product/name
192
Select a product
Go
Select a product
Widget
Gadget
Gizmo
194
Product
Features
Widget
Select a feature
GO
Gadget
Select a feature
GO
Gizmo
Select a feature
GO
Select a feature
Low maintenance
Trouble free

Fig. 2A(Prior Art)

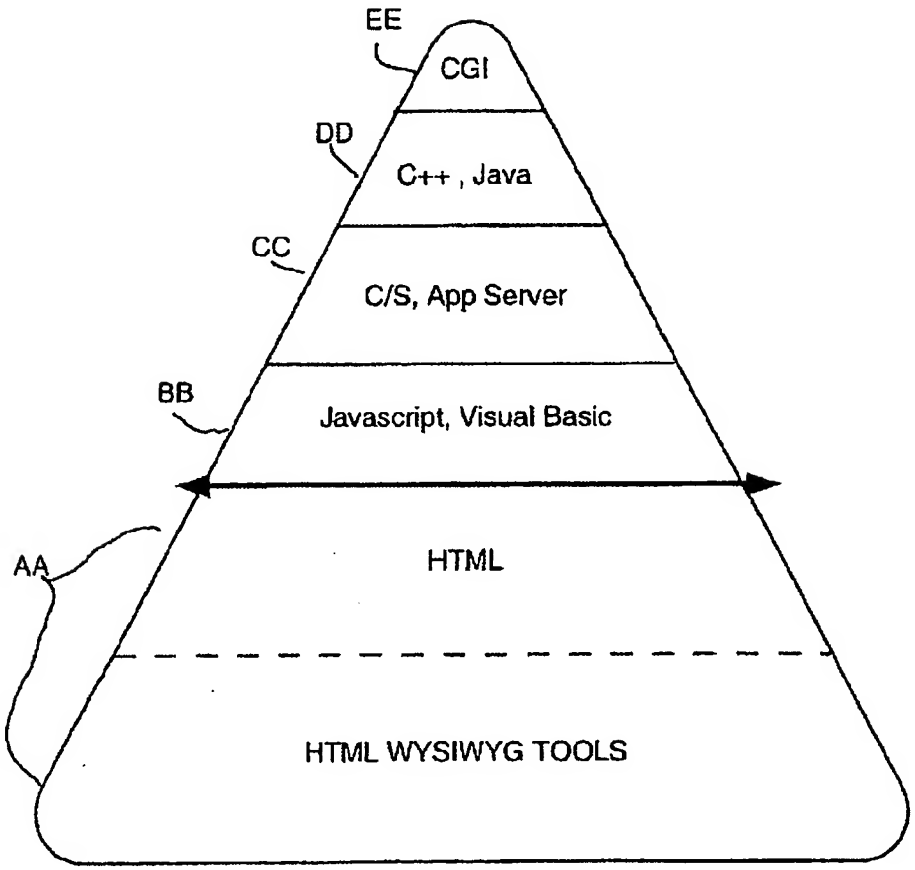


Fig. 2B (Prior Art)

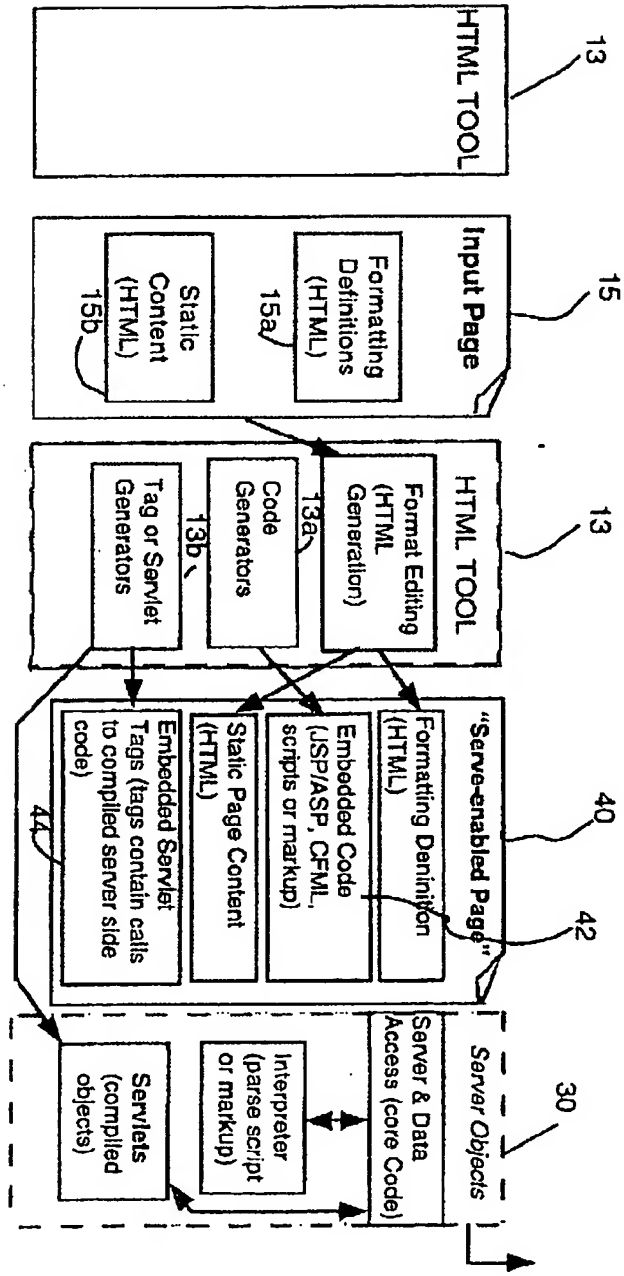


Fig. 3 (Prior Art)

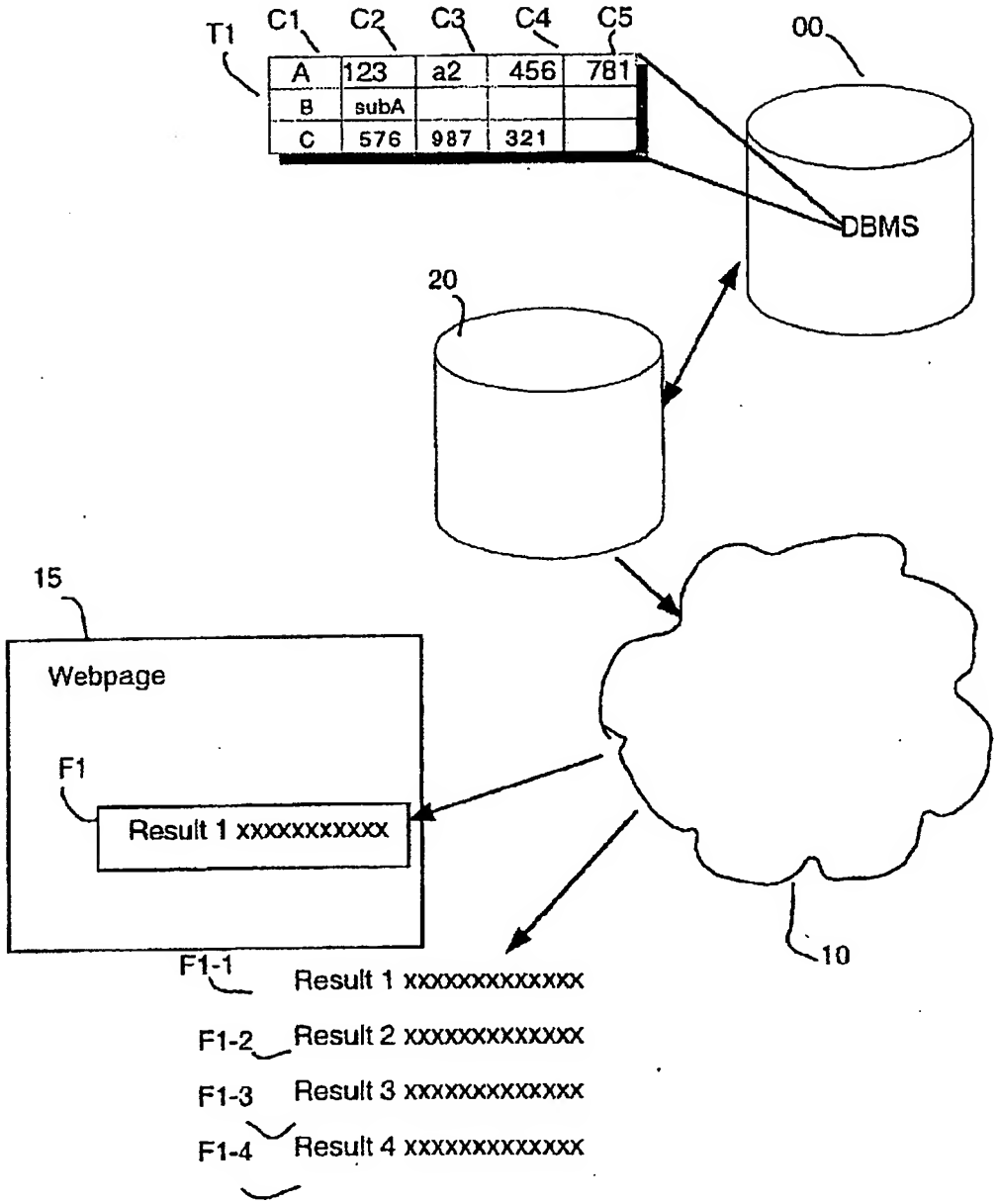


Fig. 4A (Prior Art)

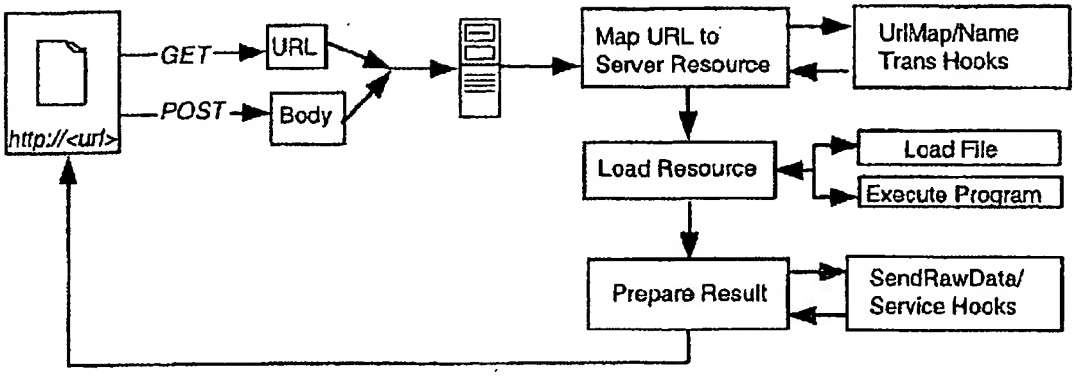
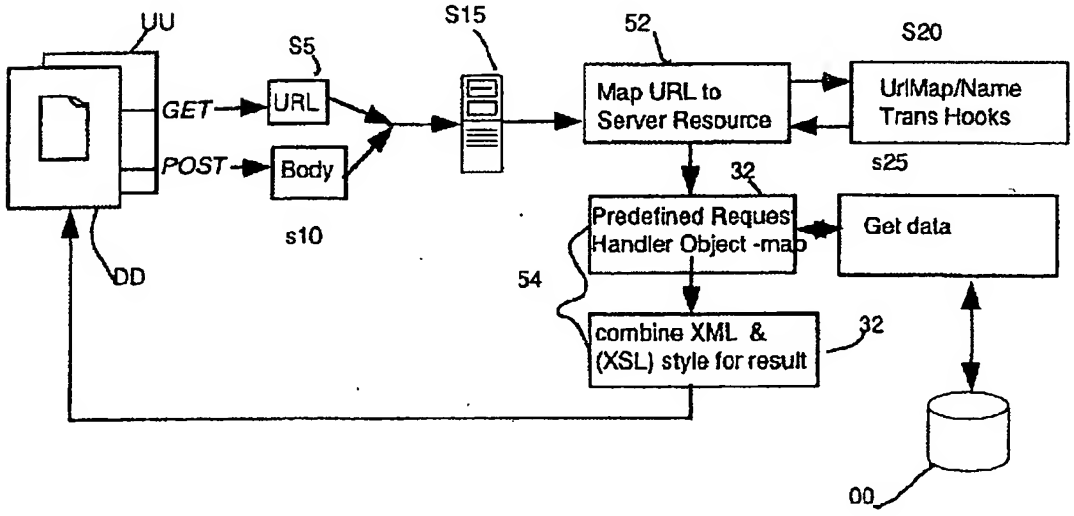


Fig. 4B



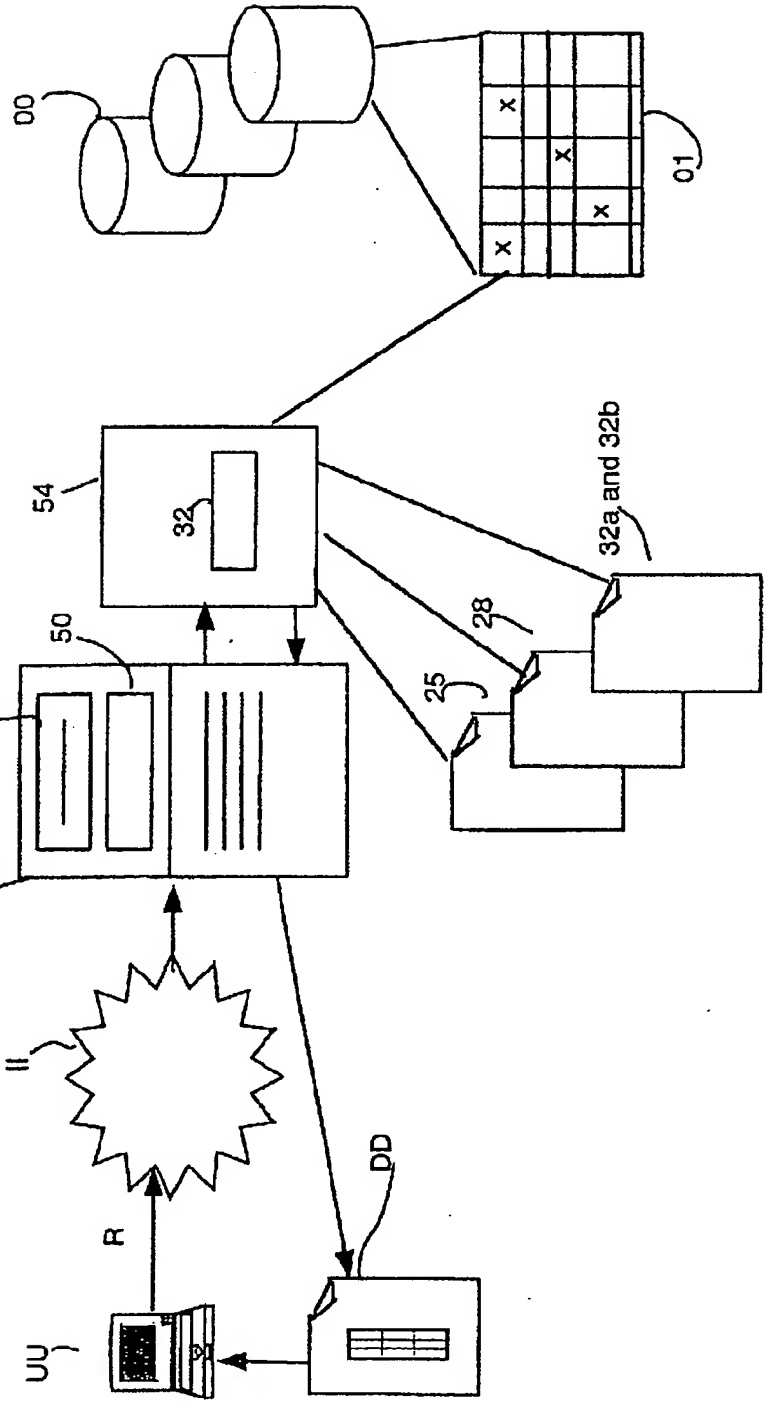


Fig. 4c

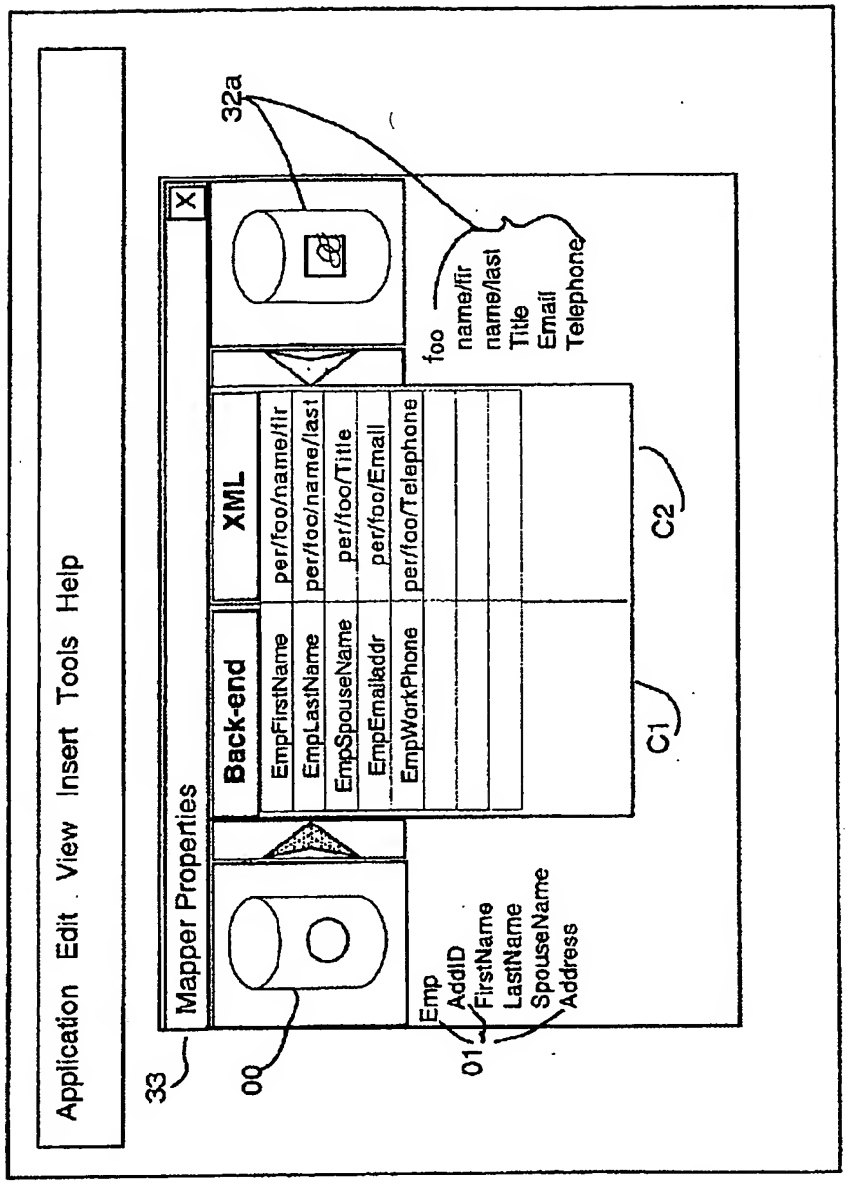

Application Edit View Insert Tools Help
Mapper Properties
Back-end
XML
EmpFirstName
EmpLastName
EmpSpouseName
EmpEmailaddr
EmpWorkPhone
per/foo/name/fir
per/foo/name/last
per/foo/Title
per/foo/Email
per/foo/Telephone
32a
foo
name/fir
name/last
Title
Email
Telephone
C2
C1
33
00
01
Emp
AddID
FirstName
LastName
SpouseName
Address

Fig. 4D

Fig. 5

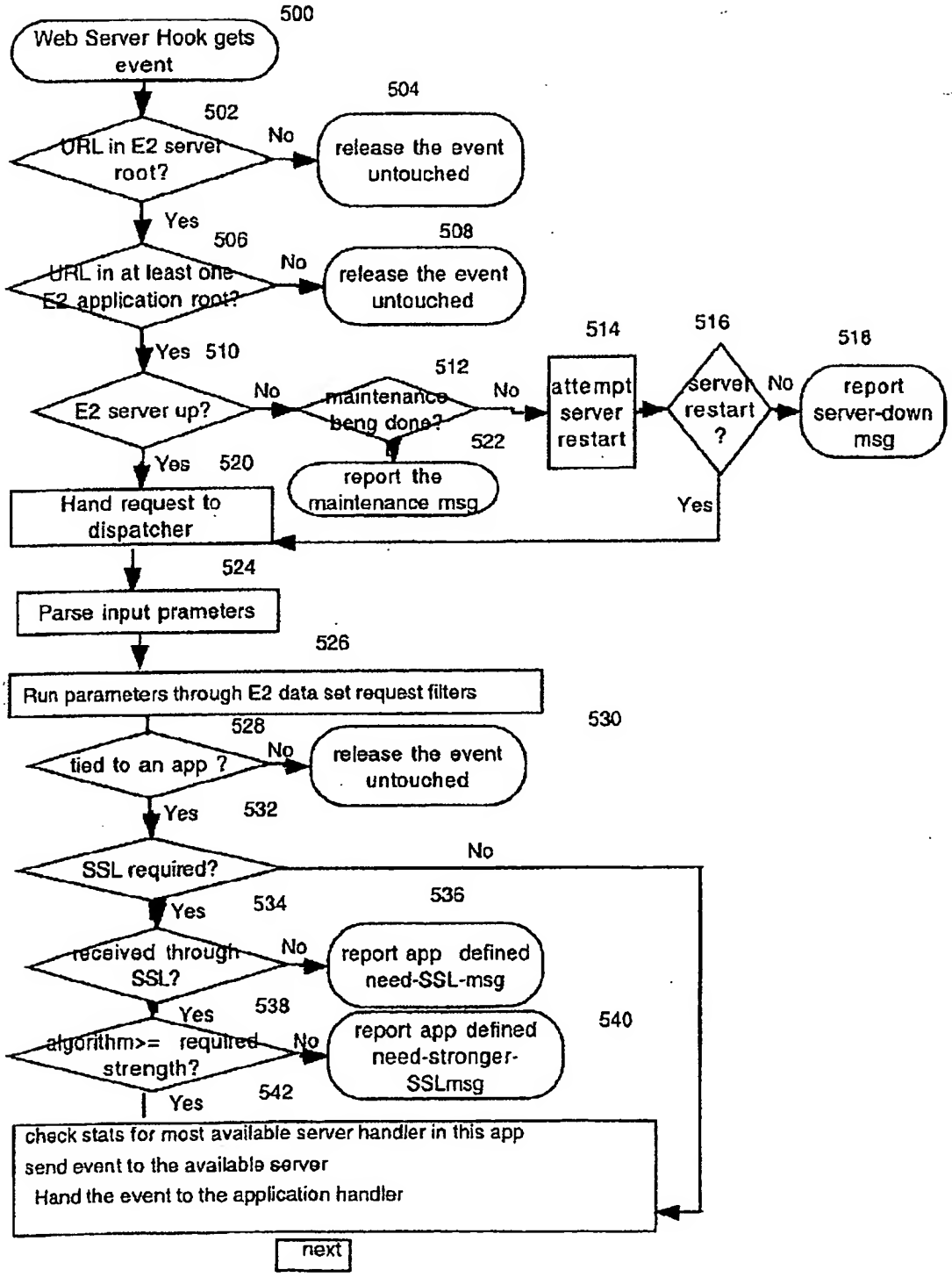

500
Web Server Hook gets event
502
URL in E2 server root?
No
504
release the event untouched
Yes
506
URL in at least one E2 application root?
No
508
release the event untouched
Yes 510
E2 server up?
No
512
maintenance being done?
No
514
attempt server restart
516
server restart ?
No
518
report server-down msg
Yes
522
report the maintenance msg
Yes 520
Hand request to dispatcher
524
Parse input prameters
526
Run parameters through E2 data set request filters
528
tied to an app ?
No
530
release the event untouched
Yes
532
SSL required?
No
Yes
534
536
received through SSL?
No
report app defined need-SSL-msg
Yes
538
algorithm>= required strength?
No
report app defined need-stronger-SSLmsg
540
Yes
542
check stats for most available server handler in this app
send event to the available server
Hand the event to the application handler
next

Fig. 6

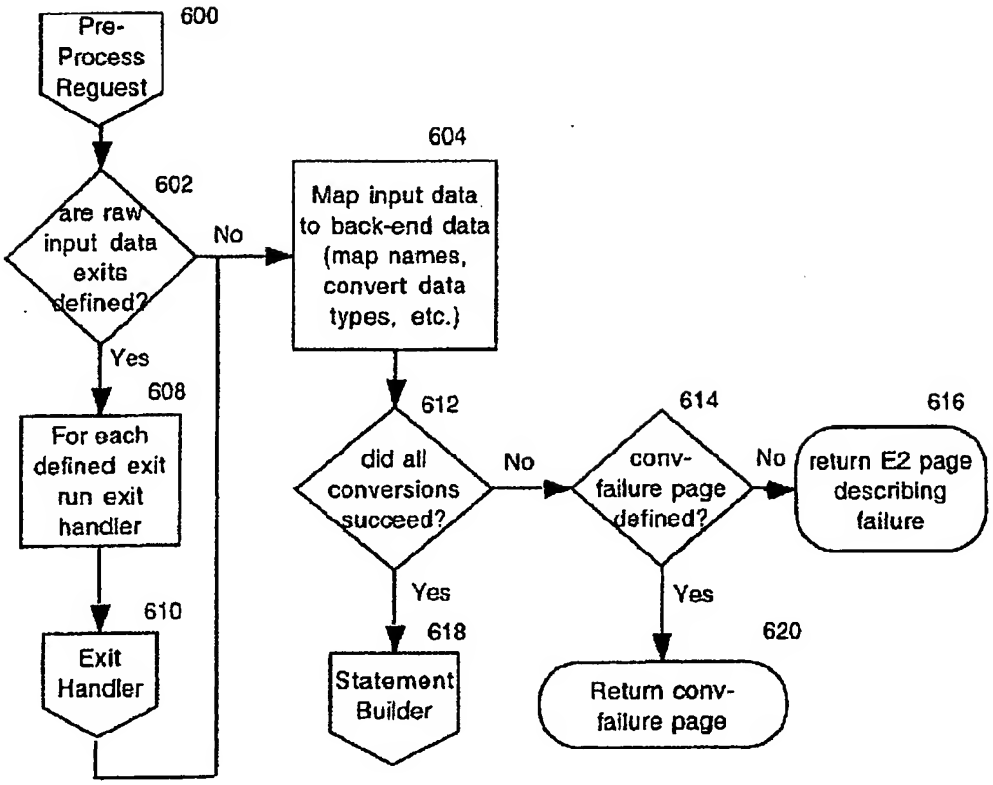

Pre-Process Reguest
600
are raw input data exits defined?
602
No
Yes
For each defined exit run exit handler
608
Exit Handler
610
Map input data to back-end data (map names, convert data types, etc.)
604
did all conversions succeed?
612
No
Yes
Statement Builder
618
conv-failure page defined?
614
No
Yes
return E2 page describing failure
616
Return conv-failure page
620

Fig. 7

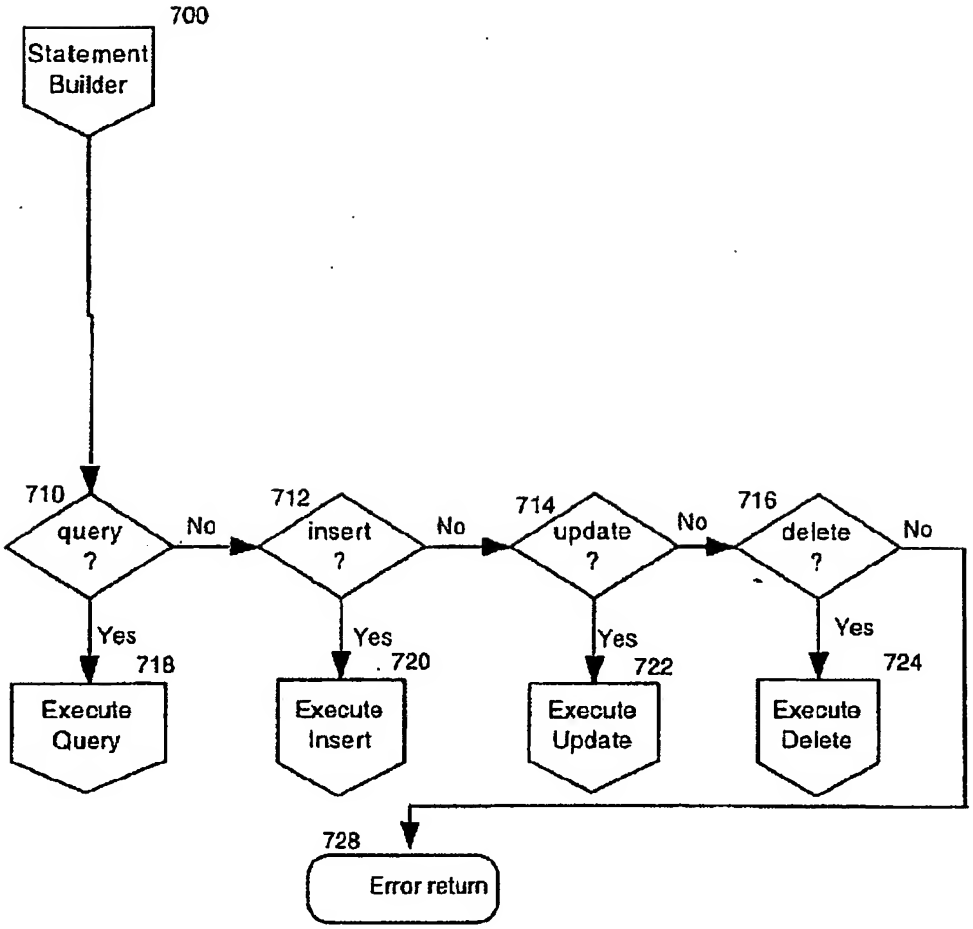

700
Statement Builder
710
query ?
No
712
insert ?
No
714
update ?
No
716
delete ?
No
Yes
718
Execute Query
Yes
720
Execute Insert
Yes
722
Execute Update
Yes
724
Execute Delete
728
Error return

Fig. 8

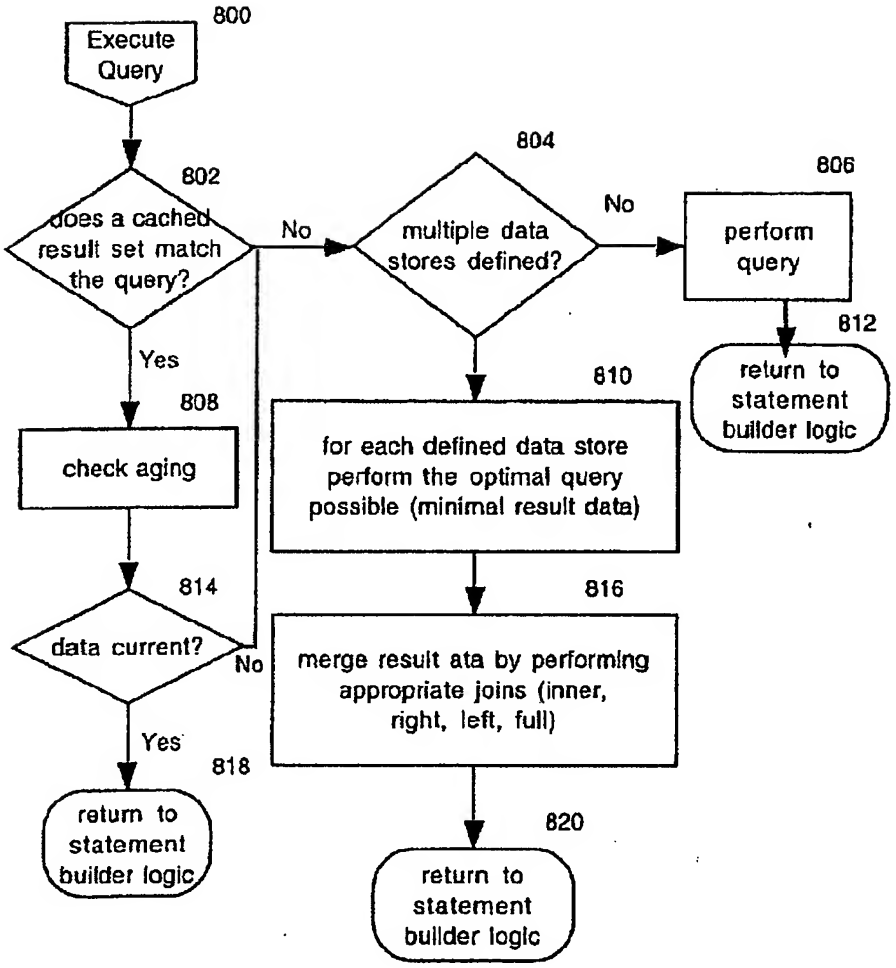

800
Execute Query
802
does a cached result set match the query?
No
804
multiple data stores defined?
No
806
perform query
812
return to statement builder logic
Yes
808
check aging
810
for each defined data store perform the optimal query possible (minimal result data)
814
data current?
No
816
merge result ata by performing appropriate joins (inner, right, left, full)
Yes
818
return to statement builder logic
820
return to statement builder logic

Fig. 9

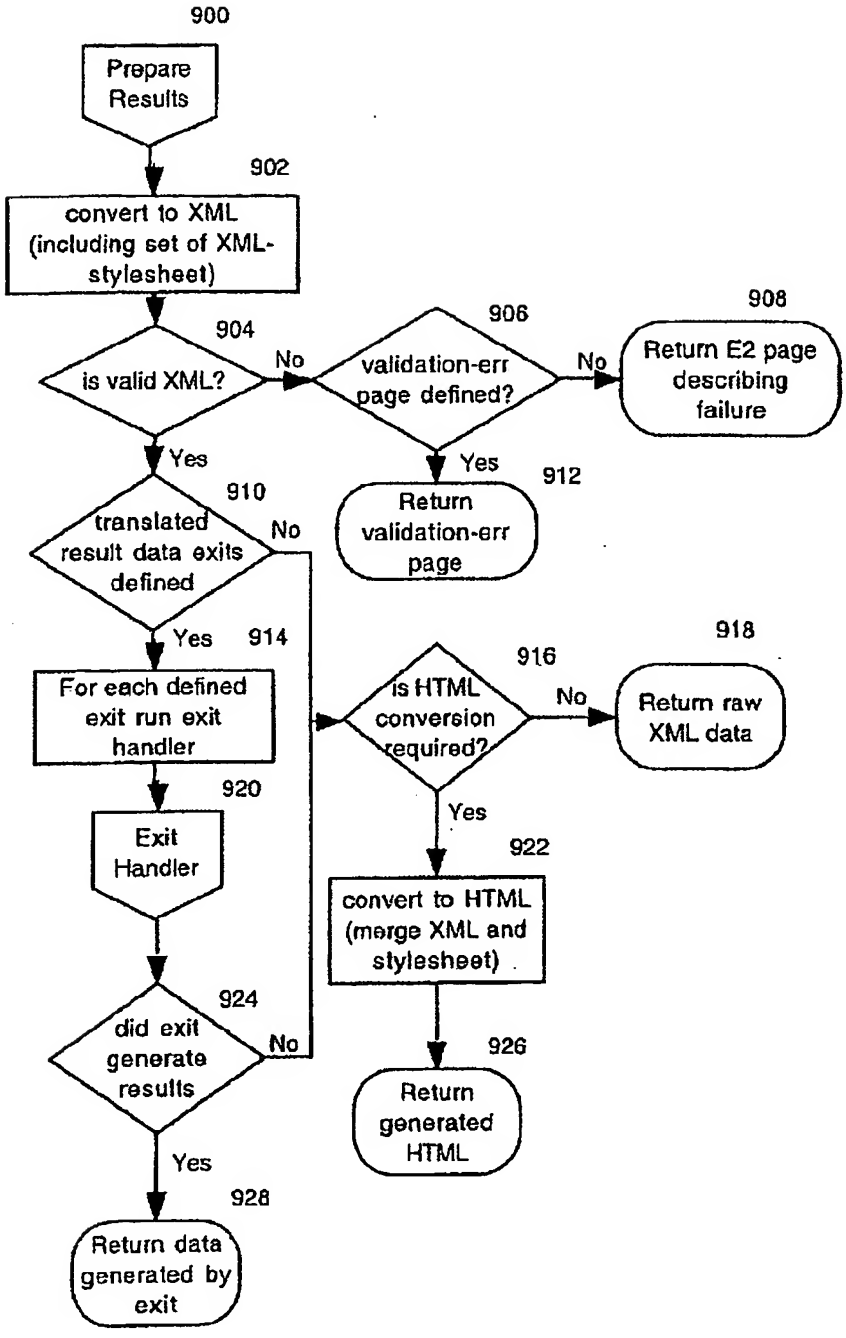

900
Prepare Results
902
convert to XML (including set of XML-stylesheet)
904
is valid XML?
No
906
validation-err page defined?
No
908
Return E2 page describing failure
Yes
Yes
912
Return validation-err page
910
translated result data exits defined
No
Yes
914
For each defined exit run exit handler
916
is HTML conversion required?
No
918
Return raw XML data
Yes
920
Exit Handler
922
convert to HTML (merge XML and stylesheet)
924
did exit generate results
No
926
Return generated HTML
Yes
928
Return data generated by exit

Fig. 10

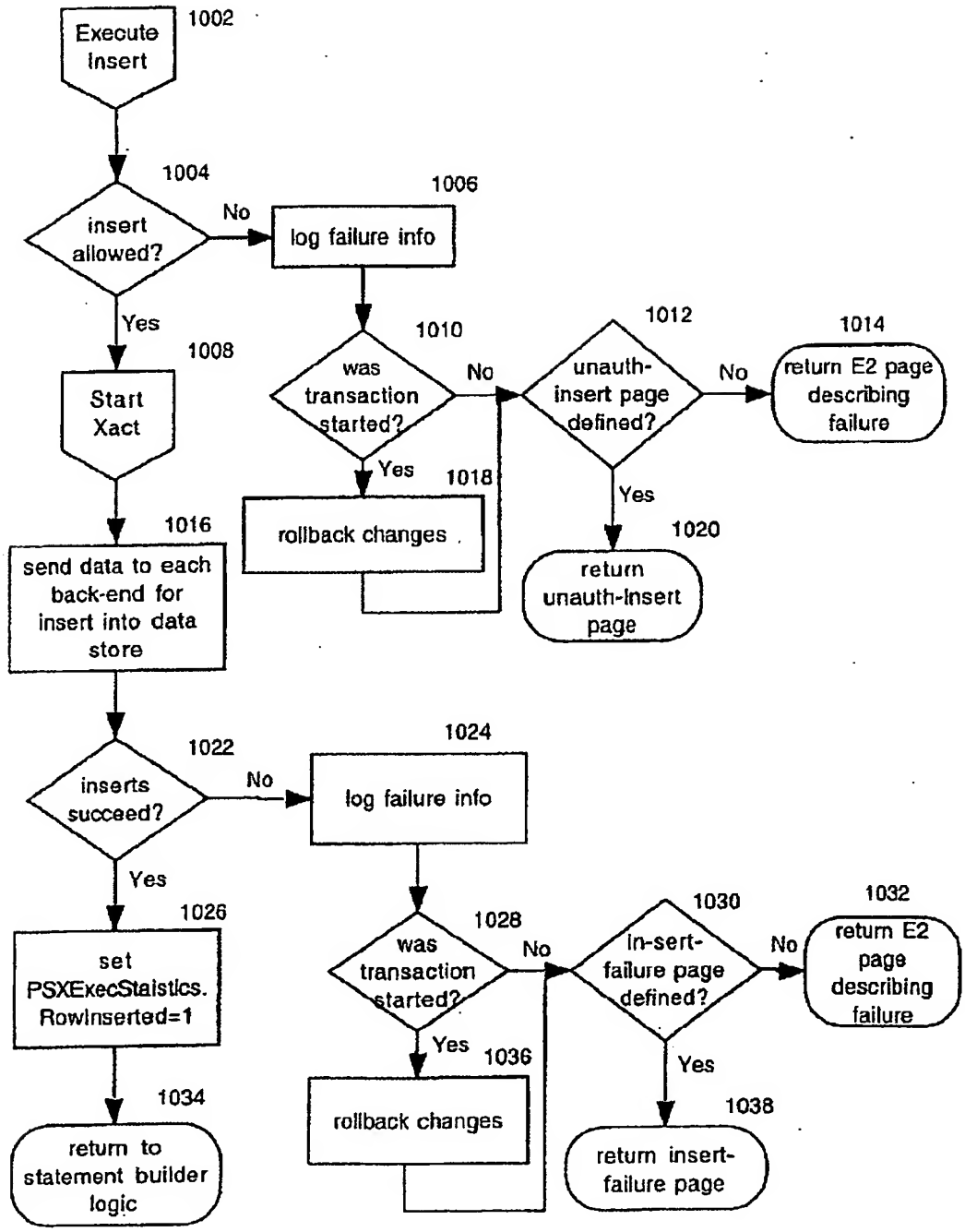

Execute Insert
1002
1004
insert allowed?
No
1006
log failure info
Yes
1008
Start Xact
1010
was transaction started?
No
1012
unauth-insert page defined?
No
1014
return E2 page describing failure
Yes
1018
rollback changes
Yes
1020
return unauth-Insert page
1016
send data to each back-end for insert into data store
1022
inserts succeed?
No
1024
log failure info
Yes
1026
set PSXExecStaistics. RowInserted=1
1028
was transaction started?
No
1030
In-sert-failure page defined?
No
1032
return E2 page describing failure
Yes
1036
rollback changes
Yes
1038
return insert-failure page
1034
return to statement builder logic

Fig. 11

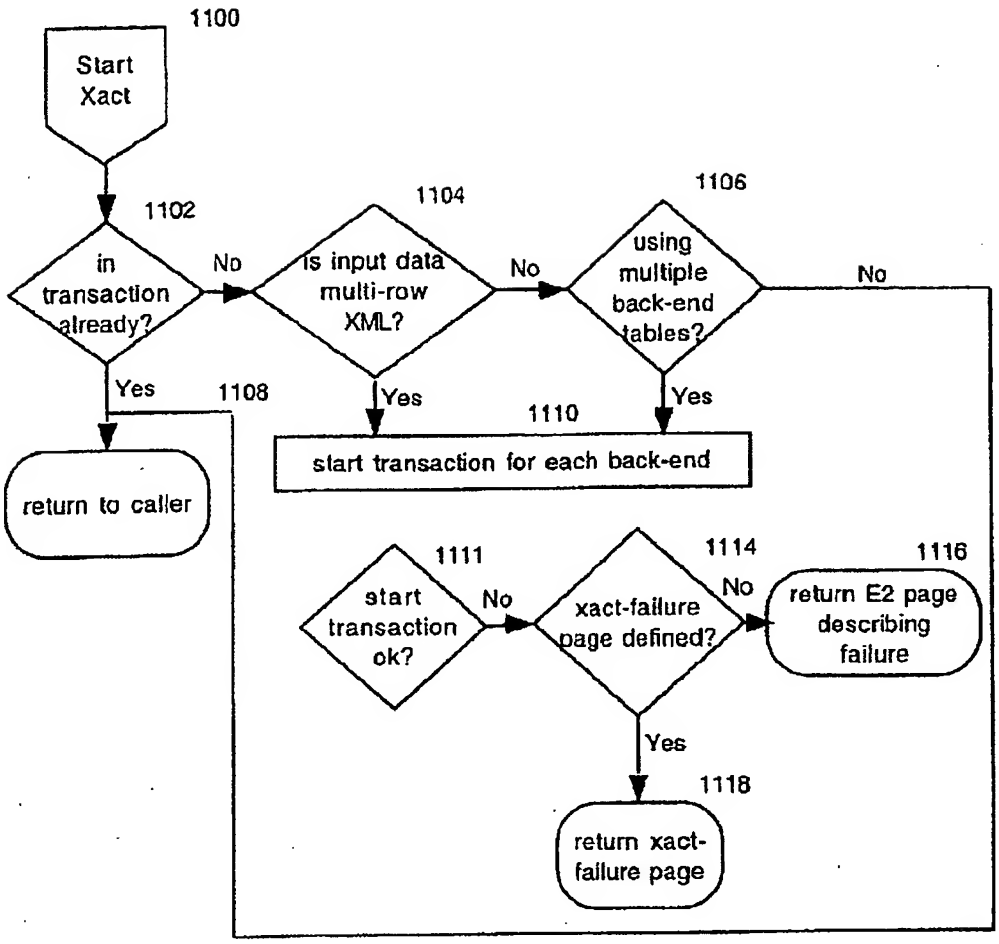

1100
Start Xact
1102
in transaction already?
No
1104
is input data multi-row XML?
No
1106
using multiple back-end tables?
No
Yes
1108
Yes
1110
Yes
start transaction for each back-end
return to caller
1111
start transaction ok?
No
1114
xact-failure page defined?
No
1116
return E2 page describing failure
Yes
1118
return xact-failure page

Fig. 12

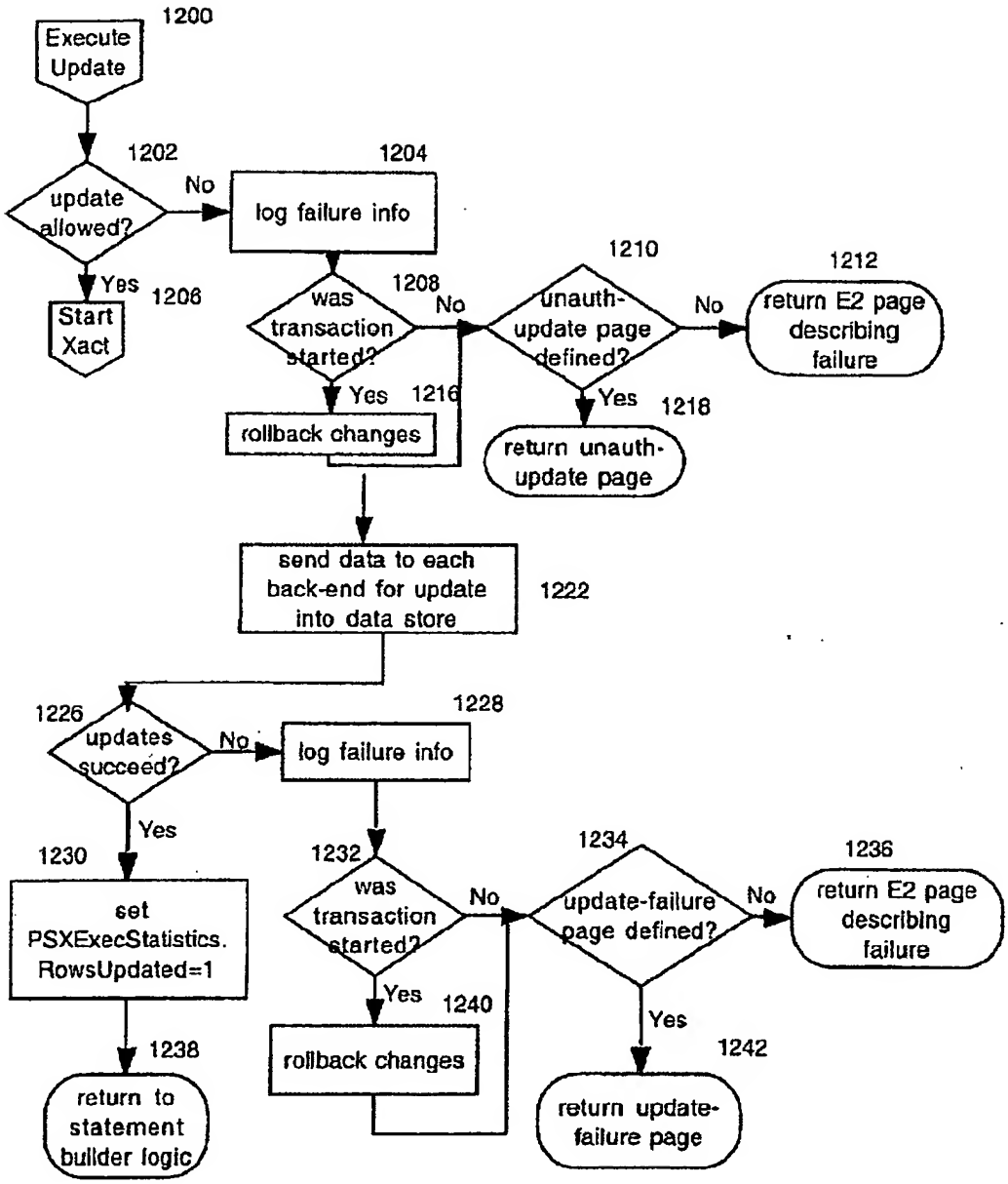

1200
Execute Update
1202
update allowed?
No
1204
log failure info
Yes
1206
Start Xact
1208
was transaction started?
No
1210
unauth-update page defined?
No
1212
return E2 page describing failure
Yes
1216
rollback changes
Yes
1218
return unauth-update page
send data to each back-end for update into data store
1222
1226
updates succeed?
No
1228
log failure info
Yes
1230
set PSXExecStatistics. RowsUpdated=1
1232
was transaction started?
No
1234
update-failure page defined?
No
1236
return E2 page describing failure
Yes
1240
rollback changes
Yes
1242
return update-failure page
1238
return to statement builder logic

Fig. 13

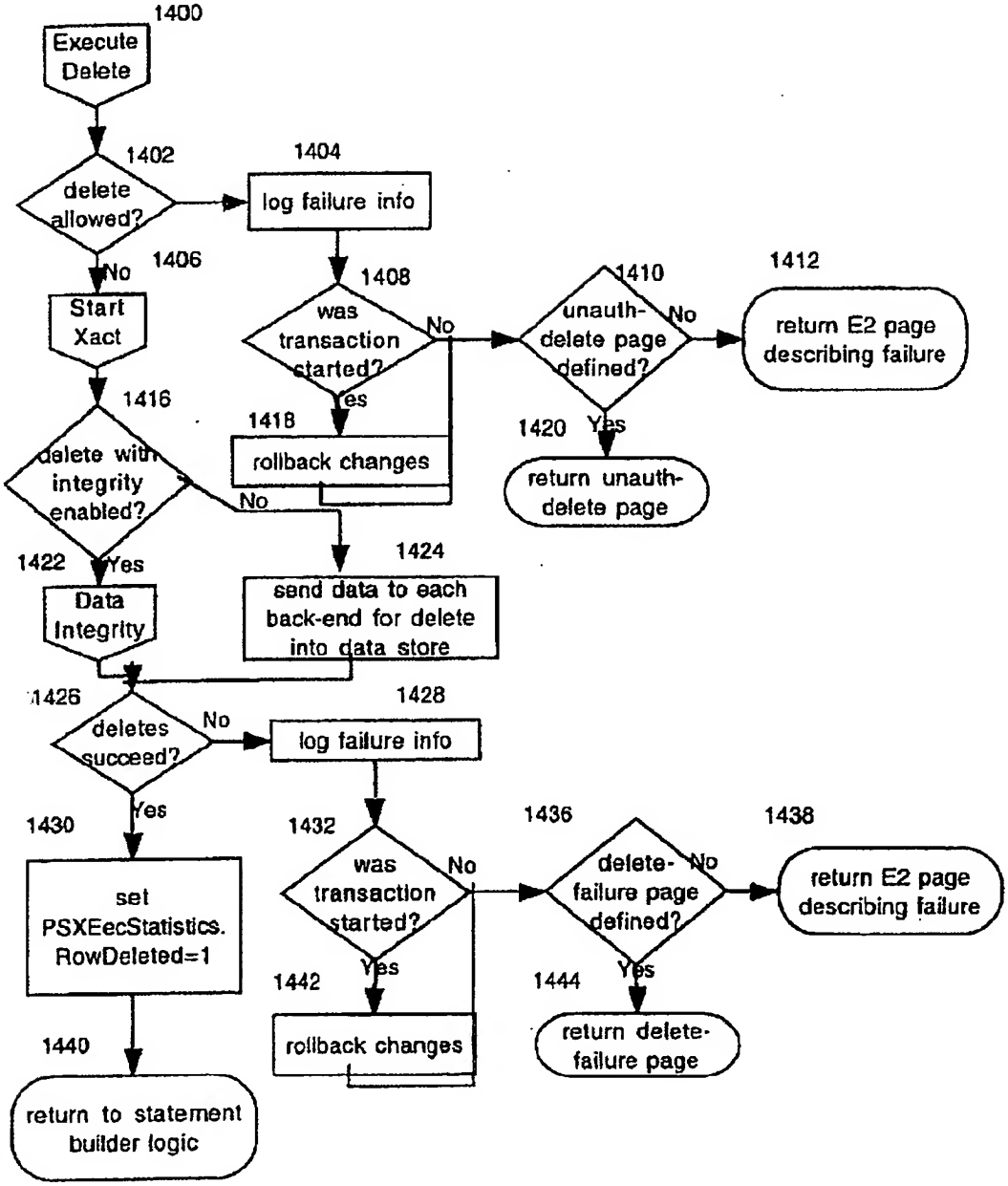

1400
Execute Delete
1402
delete allowed?
No
1404
log failure info
1406
Start Xact
1408
was transaction started?
No
Yes
1410
unauth-delete page defined?
No
1412
return E2 page describing failure
1416
delete with integrity enabled?
No
Yes
1418
rollback changes
1420
Yes
return unauth-delete page
1422
Data Integrity
1424
send data to each back-end for delete into data store
1426
deletes succeed?
No
1428
log failure info
Yes
1430
set PSXEecStatistics. RowDeleted=1
1432
was transaction started?
No
Yes
1436
delete-failure page defined?
No
1438
return E2 page describing failure
1442
rollback changes
1444
Yes
return delete-failure page
1440
return to statement builder logic

Fig. 14

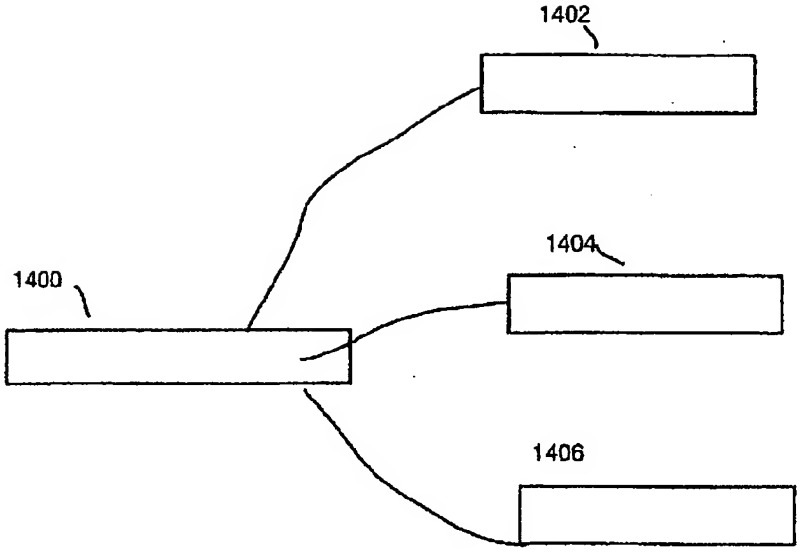
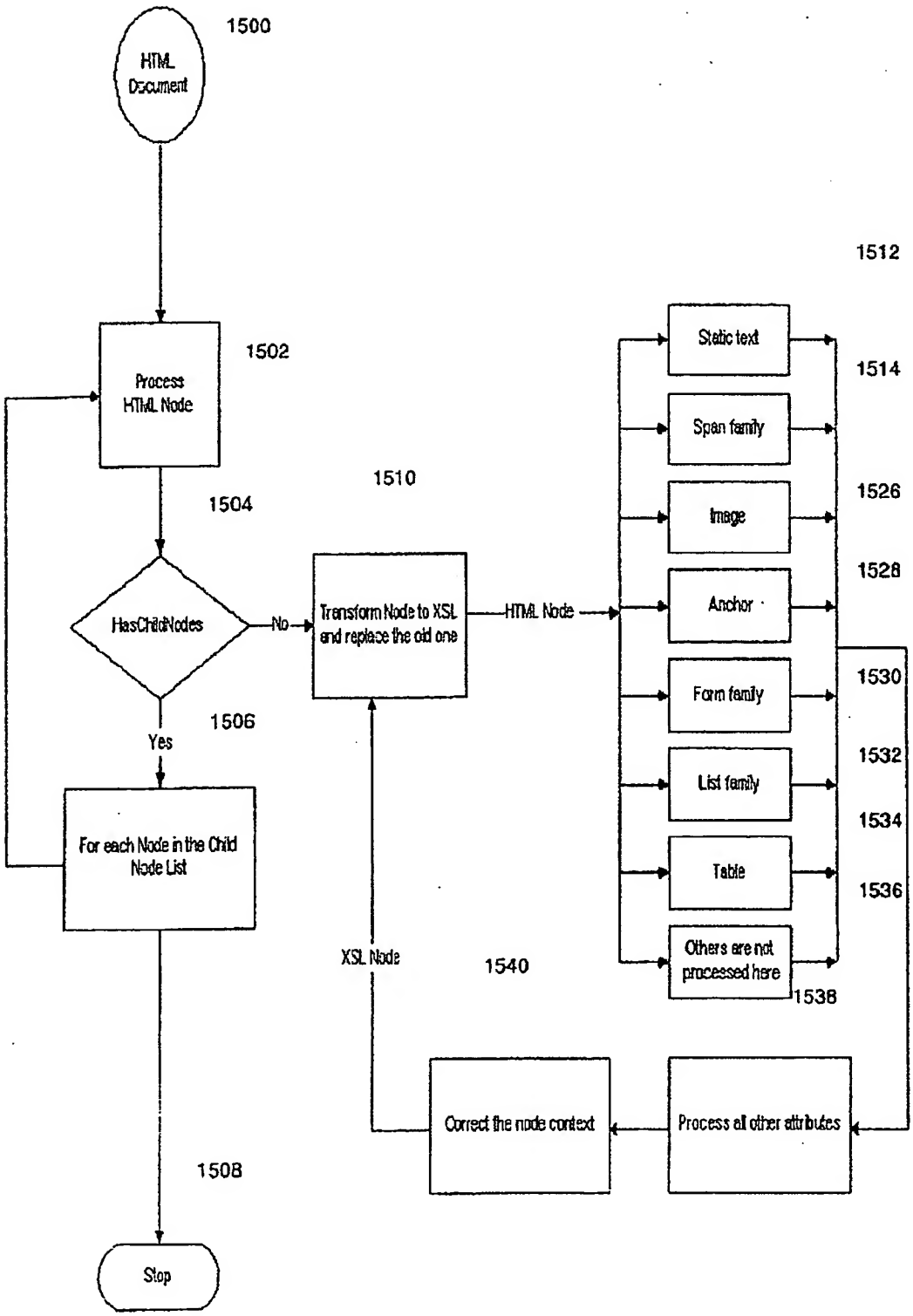


Fig. 15


1500
HTML Document
Process HTML Node
1502
1504
HasChildNodes
No
Yes
1506
For each Node in the Child Node List
1508
Stop
1510
Transform Node to XSL and replace the old one
HTML Node
XSL Node
1540
1512
Static text
1514
Span family
Image
1526
Anchor
1528
Form family
1530
List family
1532
Table
1534
1536
Others are not processed here
1538
Process all other attributes
Correct the node context

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/24695

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 13/00, 15/00

US CL : 345/333; 707/513; 709/217

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 345/333, 516; 707/513, 515, 516, 517, 520, 522; 709/217, 218, 219, 201 202, 203

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

None

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST: USPAT file.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	US 6,012,071 A (KRISHNA ET AL) 04 January 2000, col. 4, line 10 - col. 5, line 5; col. 6, lines 17-59; col. 7, line 47 - col. 9, line 3.	1-16
A,P	US 6,035,119 A (MASSENA et al) 07 March 2000, col. 2, line 44 - col. 3, line 58.	1-16
A,P	US 6,067,559 A(ALLARD et al) 23 May 2000, col. 4, line 26 - col. 5, line 9	1-16
A,P	US 5,983,267 A (SHKLAR et al.) 09 November 1999, col. 1, line 60 - col. 2, line 41.	1-16
A	US 5,748,188 A (HU et al) 05 May 1998, col. 1, lines 51-64.	1-16

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
30 OCTOBER 2000	06 DEC 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer HUYNH-BA
Facsimile No. (703) 305-3230	Telephone No. (703) 305-9794

Form PCT/ISA/210 (second sheet) (July 1998)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/24695

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,870,746 A (KNUTSON et al), 09 February 1999, col. 2, lines 20-37.	1-16